Lecture Notes in Computer Science 2678
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Wil van der Aalst Arthur ter Hofstede
Mathias Weske (Eds.)

Business Process Management

International Conference, BPM 2003
Eindhoven, The Netherlands, June 26-27, 2003
Proceedings

Springer

Volume Editors

Wil M.P. van der Aalst
Eindhoven University of Technology
Department of Information and Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
E-mail: w.m.p.v.d.aalst@tm.tue.nl

Arthur ter Hofstede
Queensland University of Technology
Centre for Information Technology Innovation
GPO Box 2434, Brisbane Qld 4001, Australia
E-mail: a.terhofstede@qut.edu.au

Mathias Weske
Potsdam University
Hasso Plattner Institute for Software Systems Engineering
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
E-mail: mathias.weske@hpi.uni-potsdam.de

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): H.3.5, H.4.1, H.5.3, K.4.3, K.4.4, K.6, J.1

ISSN 0302-9743
ISBN 3-540-40318-3 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin GmbH
Printed on acid-free paper SPIN: 10927564 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the International Conference on Business Process Management: On the Application of Formal Methods to "Process-Aware" Information Systems (BPM 2003), Eindhoven, The Netherlands, June 26-27, 2003. The conference was held in conjunction with the 24th International Conference on Application and Theory of Petri Nets (ICATPN 2003) and followed by the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003). The aim of co-locating these conferences is to stimulate interaction between researchers and practitioners working on formal methods and business process management.

BPM 2003 was organized by the Information Systems (IS) and Information & Technology (I&T) research groups of the Technische Universiteit Eindhoven (TU/e), Eindhoven, The Netherlands. We would like to thank the members of the program committee (see next page) and the reviewers for their efforts in selecting the papers. We received contributions from 27 countries. In total 77 papers were submitted. Of these papers, 25 papers were accepted for publication in these proceedings and presentation at the conference. In addition to the regular presentations, invited lectures were given by John Hoogland, Skip Ellis, Jon Pyke, and Kees van Hee.

BPM 2003 and ICATPN 2003 were supported by the following institutions and organizations: KNAW, NWO, Deloitte & Touche, Pallas Athena, BETA, Philips, Gemeente Eindhoven, Sodexho, OCE, FileNet, TU/e, SIGPAM, EMISA, and Atos Origin.

We would like to acknowledge the support of the people in the IS and I&T research groups involved in the organization of ICATPN 2003 and BPM 2003. Specifically we would like to thank Eric Verbeek for his technical support and Hajo Reijers for chairing the organization committee. Finally, we would like to mention the excellent co-operation with Springer-Verlag during the preparation of this volume.

April 2003

Wil van der Aalst
Arthur ter Hofstede
Mathias Weske

Organizing Committee

Wil van der Aalst
Cecile Brouwer
Kees van Hee
Reinier Post
Hajo Reijers (chair)
Natalia Sidorova
Eric Verbeek
Ineke Withagen

Tool Demonstration

Eric Verbeek (chair)

Program Committee

Wil van der Aalst, The Netherlands (co-chair)
Fabio Casati, USA
Jörg Desel, Germany
Susanna Donatelli, Italy
Jan Dietz, The Netherlands
Dimitrios Georgakopoulos, USA
Kees van Hee, The Netherlands
Arthur ter Hofstede, Australia (co-chair)
Geert-Jan Houben, The Netherlands
Stefan Jablonski, Germany
Akhil Kumar, USA
Ronald Lee, USA
Dan Marinescu, USA
Giorgio De Michelis, Italy
Andreas Oberweis, Germany
Michael Rosemann, Australia
Edward Stohr, USA
Gottfried Vossen, Germany
Mathias Weske, Germany (co-chair)
Leon Zhao, USA

Referees

Ad Aerts
Alessandra Agostini
Donald Baker
Giuseppe Berio
Henry H. Bi
Ladislau Boloni
Malu Castellanos
Andrzej Cichocki
Isaac Council
Alexandra Cristea
Flavio De Paoli
Antonio Di Leva
Piercarlo Giolito
Ed Glantz
Markus Gruene
Bart-Jan Hommes
Vera Hsu
Jens Hündling
Jens Lechtenbörger
Kirsten Lenz
Rong Liu
Marco Loregian
Therani Madhusudan
Christian Meiler
Marco von Mevius
Marian Nodine
Natalia Sidorova
Carla Simone

Shuang Sun
Jinesh Varia
Marc Voorhoeve
Loucif Zerguini
Zuopeng Zhang
Jerry Wang
Te-Wei Wang
Peter Westerkamp

Table of Contents

Introduction

Full Papers

Tool Papers

Business Process Management: A Survey

Wil M.P. van der Aalst[1], Arthur H.M. ter Hofstede[2], and Mathias Weske[3]

[1] Department of Technology Management
Eindhoven University of Technology, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl
[2] Centre for Information Technology Innovation
Queensland University of Technology, Australia
a.terhofstede@qut.edu.au
[3] Hasso Plattner Institute for Software Systems Engineering
University of Potsdam, Potsdam, Germany
mathias.weske@hpi.uni-potsdam.de

Abstract. Business Process Management (BPM) includes methods, techniques, and tools to support the design, enactment, management, and analysis of operational business processes. It can be considered as an extension of classical Workflow Management (WFM) systems and approaches. Although the practical relevance of BPM is undisputed, a clear definition of BPM and related acronyms such as BAM, BPA, and STP are missing. Moreover, a clear scientific foundation is missing. In this paper, we try to demystify the acronyms in this domain, describe the state-of-the-art technology, and argue that BPM could benefit from formal methods/languages (cf. Petri nets, process algebras, etc.).

Keywords: Business Process Management, Workflow Management, Formal Methods.

1 Introduction

This volume of Springer Lecture Notes in Computer Science is devoted to the "Conference on Business Process Management: On the Application of Formal Methods to Process-Aware Information Systems" taking place in Eindhoven (The Netherlands) in June 2003. To put the contributions to this conference into perspective, we discuss the ideas, technology, and foundations hidden behind acronyms like WFM, BPM, BAM, BPA, STP, etc. The goal of this paper is to provide an overview of the scientific and practical issues in the context of business process management systems. This way we hope to trigger researchers and practitioners to address the challenges in this domain. The definition of a business process management system used throughout this paper is: *a generic software system that is driven by explicit process designs to enact and manage operational business processes*. The system should be process-aware and generic in the sense that it is possible to modify the processes it supports. The process designs are often graphical and the focus is on structured processes that need to handle many cases.

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 1–12, 2003.

To show the relevance of business process management systems, it is interesting to put them in a historical perspective. Consider Figure 1, which shows some of the ongoing trends in information systems [3]. This figure shows that today's information systems consist of a number of layers. The center is formed by the operating system, i.e., the software that makes the hardware work. The second layer consists of generic applications that can be used in a wide range of enterprises. Moreover, these applications are typically used within multiple departments within the same enterprise. Examples of such generic applications are a database management system, a text editor, and a spreadsheet program. The third layer consists of domain specific applications. These applications are only used within specific types of enterprises and departments. Examples are decision support systems for vehicle routing, call center software, and human resource management software. The fourth layer consists of tailor-made applications. These applications are developed for specific organizations.

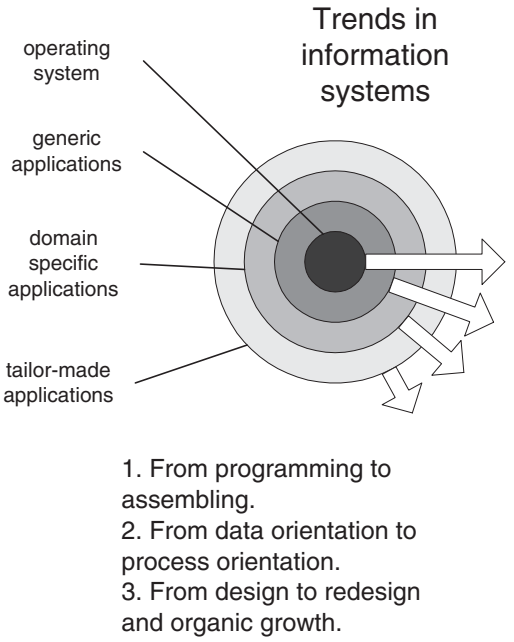

Fig. 1. Trends relevant for business process management [3].

In the sixties the second and third layer were missing. Information systems were built on top of a small operating system with limited functionality. Since no generic nor domain specific software was available, these systems mainly consisted of tailor-made applications. Since then, the second and third layer have developed and the ongoing trend is that the four circles are increasing in size, i.e., they are moving to the outside while absorbing new functionality. Today's

operating systems offer much more functionality. Database management systems that reside in the second layer offer functionality which used to be in tailor-made applications. As a result of this trend, the emphasis shifted from programming to assembling of complex software systems. The challenge no longer is the coding of individual modules but orchestrating and gluing together pieces of software from each of the four layers.

Another trend is the shift from data to processes. The seventies and eighties were dominated by data-driven approaches. The focus of information technology was on storing and retrieving information and as a result data modeling was the starting point for building an information system. The modeling of business processes was often neglected and processes had to adapt to information technology. Management trends such as business process reengineering illustrate the increased emphasis on processes. As a result, system engineers are resorting to a more process driven approach.

The last trend we would like to mention is the shift from carefully planned designs to redesign and organic growth. Due to the omnipresence of the Internet and its standards, information systems change on-the-fly. As a result, fewer systems are built from scratch. In many cases existing applications are partly used in the new system. Although component-based software development still has its problems, the goal is clear and it is easy to see that software development has become more dynamic.

The trends shown in Figure 1 provide a historical context for business process management systems. Business process management systems are either separate applications residing in the second layer or are integrated components in the domain specific applications, i.e., the third layer. Notable examples of business process management systems residing in the second layer are workflow management systems [6,19,22,23,24] such as Staffware, MQSeries, and COSA, and case handling systems such as FLOWer. Note that leading enterprise resource planning systems populating the third layer also offer a workflow management module. The workflow engines of SAP, Baan, PeopleSoft, Oracle, and JD Edwards can be considered as integrated business process management systems. The idea to isolate the management of business processes in a separate component is consistent with the three trends identified. Business process management systems can be used to avoid hard-coding the work processes into tailor-made applications and thus support the shift from programming to assembling. Moreover, process orientation, redesign, and organic growth are supported. For example, today's workflow management systems can be used to integrate existing applications and support process change by merely changing the workflow diagram. Isolating the management of business processes in a separate component is also consistent with recent developments in the domain of web services: Web services composition languages such as BPEL4WS, BPML, WSCI, XLANG, and WSFL can be used to glue services defined using WSDL together.

An interesting starting point from a scientific perspective is the early work on office information systems. In the seventies, people like Skip Ellis [13], Anatol Holt [17], and Michael Zisman [28] already worked on so-called office informa-

tion systems, which were driven by explicit process models. It is interesting to see that the three pioneers in this area independently used Petri-net variants to model office procedures. During the seventies and eighties there was great optimism about the applicability of office information systems. Unfortunately, few applications succeeded. As a result of these experiences, both the application of this technology and research almost stopped for a decade. Consequently, hardly any advances were made in the eighties. In the nineties, there again was a huge interest in these systems. The number of workflow management systems developed in the past decade and the many papers on workflow technology illustrate the revival of office information systems. Today workflow management systems are readily available [22]. However, their application is still limited to specific industries such as banking and insurance. As was indicated by Skip Ellis it is important to learn from these ups and downs [14]. The failures in the eighties can be explained by both technical and conceptual problems. In the eighties, networks were slow or not present at all, there were no suitable graphical interfaces, and proper development software was missing. However, there were also more fundamental problems: a unified way of modeling processes was missing and the systems were too rigid to be used by people in the workplace. Most of the technical problems have been resolved by now. However, the more conceptual problems remain. Good standards for business process modeling are still missing and even today's workflow management systems enforce unnecessary constraints on the process logic (e.g., processes are made more sequential).

2 Business Process Management Demystified

Many people consider Business Process Management (BPM) to be the "next step" after the workflow wave of the nineties. Therefore, we use workflow terminology to define BPM. The Workflow Management Coalition (WfMC) defines workflow as: "The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." [22]. A Workflow Management System (WFMS) is defined as: "A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications." [22]. Note that both definitions emphasize the focus on enactment, i.e., the use of software to support the execution of operational processes. In the last couple of years, many researchers and practitioners started to realize that the traditional focus on enactment is too restrictive. As a result new terms like BPM have been coined. There exist many definitions of BPM but in most cases it clearly includes Workflow Management (WFM). We define BPM as follows: *Supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information.* Note that this definition restricts BPM to operational processes, i.e., processes at the

strategic level or processes that cannot be made explicit are excluded. Note that systems supporting BPM need to be "process aware", i.e., without information about the operational processes at hand little support is possible.

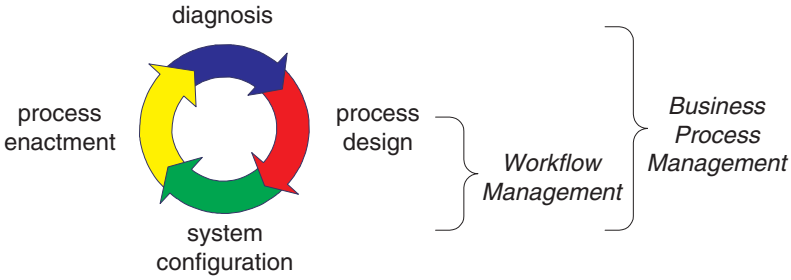

Fig. 2. The BPM lifecyle to compare Workflow Management and Business Process Management.

Figure 2 shows the relationship between WFM and BPM using the BPM lifecyle. The BPM lifecyle describes the various phases in support of operational business processes. In the design phase, the processes are (re)designed. In the configuration phase, designs are implemented by configuring a process aware information system (e.g., a WFMS). After configuration, the enactment phase starts where the operational business processes are executed using the system configured. In the diagnosis phase, the operational processes are analyzed to identify problems and to find things that can be improved. The focus of traditional workflow management (systems) is on the lower half of the BPM lifecyle. As a result there is little support for the diagnosis phase. Moreover, support in the design phase is limited to providing an editor and analysis and real design support are missing. It is remarkable that few WFM systems support simulation, verification, and validation of process designs. It is also remarkable that few systems support the collection and interpretation of real-time data. Note that most WFM systems log data on cases and tasks executed. However, no tools to support any form of diagnosis are offered by the traditional systems.

Currently, many workflow vendors are positioning their systems as BPM systems. Gartner expects the BPM market to grow and also identifies *Business Process Analysis* (BPA) as an important aspect [16]. It is expected that the BPA market will continue to grow. Note that BPA covers aspects neglected by traditional workflow products (e.g., diagnosis, simulation, etc.). *Business Activity Monitoring* (BAM) is one of the emerging areas in BPA. The goal of BAM tools is to use data logged by the information system to diagnose the operational processes. An example is the ARIS Process Performance Manager (PPM) of IDS Scheer [18]. ARIS PPM extracts information from audit trails (i.e., information logged during the execution of cases) and displays this information in a graphical way (e.g., flow times, bottlenecks, utilization, etc.). BAM also includes process

mining, i.e., extracting process models from logs [10]. BAM creates a number of scientific and practical challenges (e.g., which processes can be discovered and how much data is needed to provide useful information).

When it comes to redesigning operational processes two trends can be identified: *Straight Through Processing* (STP) and *Case Handling* (CH). STP refers to the complete automation of a business process, i.e., handling cases without human involvement. STP is often only possible if the process is redesigned. Moreover, STP is often only possible for a selected set of cases. The latter means that cases are split into two groups: (1) cases that can be handled automatically (in Dutch these cases are called "Gladde gevallen") and (2) cases that require human involvement. By separating both groups it is often possible to reduce flow time and cut costs. While STP strives for more automation, CH addresses the problem that many processes are much too variable or too complex to capture in a process diagram [4]. In CH the normal route of a case is modeled but at the same time other routes are allowed if not explicitly excluded. One way to do this is to make workflows data-driven rather than process-driven and allow for authorizations to skip or undo activities. Also the focus is on the case as a whole rather than on individual work-items distributed over work-lists.

To summarize: BPM extends the traditional WFM approach by support for the diagnosis phase (cf. BPA and BAM software) and allowing for new ways to support operational processes (cf. CH and STP). In the remainder, we focus on the scientific foundations and current technology.

3 On the Interplay between Business Process Management and Formal Methods

Business Process Models should have a formal foundation. Well-known reasons (see e.g. [1]) include: 1) formal models do not leave any scope for ambiguity, and 2) formal models increase the potential for analysis (see also e.g. [26]).

It is desirable that a Business Process Model can be understood by the various stakeholders involved in an as straightforward manner as possible. This could e.g. be achieved through the use of graphical representations. At the same time, these stakeholders should assign the same meaning to such a model, there should not be any scope for alternative interpretations. Business Process Models can be quite complex and the use of a formal language for their specification is the only sure way to guarantee that alternative interpretations are ruled out. After consensus among the stakeholders has been reached, a business process model can be deployed and if a formal language was used, its behavior can be explained in terms of the formal semantics of that specification language. As remarked in [21], the lack of a formal semantics has resulted in different interpretations by vendors of even basic control flow constructs, definitions in natural language such as provided by the Workflow Management Coalition are not precise enough.

As always, it is preferable to identify any problems in software before it is actually deployed. In the case of Business Process Models this is especially important as they may involve core business and/or complex business transactions.

To reduce the risk of costly corrections, a thorough analysis of a Business Process Model can be beneficial. Analysis of Business Process Models can also be used to investigate ways of improving processes (e.g. reducing their cost). Formal languages may have associated analysis techniques which can be used for investigating properties of specifications. These techniques can then be relied upon to provide insight into the behavior and characteristics of a Business Process Model specified in such a language.

In [1] three reasons are stated arguing the benefits of the use of Petri nets for the specification of workflows. The reasons brought forward are the fact that Petri nets are formal, have associated analysis techniques, and are state-based rather than event-based. The development of Woflan (see e.g. [25]) demonstrates that workflows specified as workflow nets [2], a subclass of Petri nets, can be analyzed in order to determine whether they are e.g. deadlock free. In the context of UML activity diagrams, tool support for verification is discussed in [15].

Through the notion of place, Petri nets provide natural support for modeling the stages in between processing. State-based patterns such as *deferred choice*, *interleaved paralled routing*, and *milestone* can therefore be specified straightforwardly. A description of these patterns can be found in [9]. Petri nets though also have some deficiencies when it comes to the specification of certain control flow dependencies (see [7]). This observation has led to the development of YAWL [8] (Yet Another Workflow Language) of which the formal semantics is specified as a transition system.

It is interesting to observe that a concept such as the deferred choice, while easily captured in terms of Petri nets, is not often supported in languages of "classical" workflow management systems (see [9]). Two recently proposed standards for web service composition, BPEL4WS and BPML, however, provide direct support for this construct (see [27]) and [5] resp.). In web services composition it is important to capture the interactions between the various services and a formalism such as the π-calculus seems to be a natural candidate to provide a formal foundation for such interactions. While it is sometimes claimed that BPML is based on the π-calculus, there does not seem to be a precise definition of this relation (note that in [12], it is stated that "there is currently no evidence that BPEL4WS is based on a formal semantics"). We believe that it is important that such relations are fully formalized.

Formally defined Business Process Modeling Languages can be compared in terms of their expressive power. For some classes of workflow modeling languages, abstractions of some existing approaches, comparative expressiveness has been studied in [21,20]. These results are in the context of a specific notion of equivalence, addressing the issue of when two workflow models can be considered expressing the same workflow. Expressiveness results give insight into what can and cannot be expressed in some approaches and more research is needed in this area as it could provide more guidance for language development.

4 Available Technology and Emerging Standards

Based on the definition of Business Process Management proposed in Section 2, a characterization of its main concepts is provided, and the technology currently available or on the horizon is discussed. Some of the key aspects of business process management already mentioned in Sections 1 and 2 are re-visited, and the current state of available technology and emerging standards are discussed.

One of the main aspects and certainly an activity typically carried out in early phases of business process management projects is the design of business processes. There is a close relationship between business process design and business process modeling, where the former refers to the overall design process involving multiple steps and the latter refers to the actual representation of the business process in terms of a business process model using a process language. To this end, the term business process modeling is used to characterize the identification and (typically rather informal) specification of the business processes at hand. This phase includes modeling of activities and their causal and temporal relationships as well as specific business rules that process executions have to comply with.

Business process modeling has a decade long tradition, and a variety of products are commercially available to support this phase, based on different process languages. Given this situation, it is not surprising that the selection of a particular product is an important step in many BPM projects, and, consequently, appropriate selection criteria have been studied extensively. Besides organizational, economical, and aspects related to the overall IT infrastructure of the enterprise at hand, the expressive power of the process language as well as interfaces to related software systems are important criteria, most prominently interfaces to process enactment systems (such as workflow management systems) and to software responsible for modeling personnel and organizational structures of the enterprise. Not only the expressive power but also a well-defined semantics of the process language deserves a central role during product selection. However, this aspect is considered only in a small number of recent business process management projects.

Business process analysis aims at investigating properties of business processes that are neither obvious nor trivial. To this end, the term analysis is used with a rather broad meaning, including for example simulation and diagnosis, verification and performance analysis. Process simulation facilitates process diagnosis in the sense that by simulating real-world cases, domain experts can acknowledge correct modeling or propose modifications of the original process model. If business process models are expressed in process languages with a clear semantics, their structural properties can be analyzed. If, for example, certain parts of processes can never be reached, an obvious modeling mistake occurred that should be fixed. While basic structural properties of process models have been studied for some time, it is remarkable that few software products actually support them. However structural analysis of process models requires a clear formal semantics of the underlying process language, which might not be present. In some products, a pragmatic approach to process modeling is preferred to a

formal one; especially if the main goal of process modeling is discussion with domain experts rather than process analysis or process enactment. However, we mention that formal semantics of process languages and intuitiveness and ease of use are no contradicting goals, and recent approaches seem to support this observation.

The next aspect of BPM and traditionally a very strong one is process enactment. However, before process enactment is discussed, we provide a coarse classification of business processes that paves the way for a discussion of different types of process enactment systems. In the early days of BPM when in the application side business process modeling and in the IT enactment side workflow management were the only options, processes with a static structure were focused. The main reason behind this obvious limitation was as follows: Modeling a process and providing infrastructure for its enactment incurs considerable effort. To provide satisfactory return on investment, a large number of individual cases have to benefit from this new technology. This type of straight-through-process is also called production workflow [23]. While there are successful workflow projects on this type of straight-through processes, this restriction of workflow technology proved fatal for applications in more dynamic environments. In some cases where traditional workflow technology was used in these advanced settings, new workflow solutions were partly circumvented or even neglected. As a response to this situation, considerable work in ad-hoc, flexible and case-based workflow was (and is being) conducted, both in academia and in industry. Recently, case handling is studied in depth as a new paradigm for supporting knowledge-intensive business processes with loose structuring. Based on the brief characterization of case handling provided above, we mention that in the case handling paradigm knowledge workers enjoy a great degree of freedom in organizing and performing their work which they are knowledgeable about. Some of the concepts of case handling are already present in commercial case handling systems.

Standardization has a long history in workflow management. Fueled by information system heterogeneity that also includes workflow management systems, organizations started to form interest groups aiming at standardizing interfaces between workflow management systems and components, with the goal of enhancing interoperability and fostering the workflow market. The most prominent organization in this context is the Workflow Management Coalition (WfMC) that was formed in 1993 and today has over 300 member organizations, including all major workflow vendors as well as workflow users and interested academia [22]. The basis of WfMC activities is the so called WfMC Reference Architecture that defines standard workflow system components interfaces. Despite the fact that all major vendors are organized in WfMC and a number of important contributions on practical workflow aspects have been made, many people feel that WfMC's ambitious goals have yet to be reached.

A more recent standardization effort in the BPM context is related to the current momentum of XML and Web services technology. Web services is a promising technology to foster interoperability between information system based –

conceptually – on the service oriented architecture paradigm [11] and – technologically – on open standards and light-weight protocols and systems. While Web services technology has not yet reached maturity level, there is considerable effort under way by literally all major software vendors. The need for standardization is clearly acknowledged in this context, and important contributions have been made. However, as sketched in Section 2, recently a trend of new standards proposals as well as merging of proposals can be experienced in the Web services context. Besides these recent developments, Web services are seen as an important infrastructure to foster business processes by composing individual Web services to represent complex processes, which can even span multiple organizations. While Web services composition is a young discipline and a number of proposals are being discussed, we currently experience what seems to be a slow consolidation of recent standardization effort around Web services composition, based on the BPEL4WS and associated proposals. However, at this point industry seems more involved in standardization than in systems design and development. While there is some controversy on these upcoming standards, it seems that at least industry goes with the flow. In any case, Web services in general and Web services composition in particular can be expected to play an important role in future business process systems technology. This will include both processes within organizations and, more strongly, between organizations.

5 Conclusion

This paper provides an overview of Business Process Management (BPM) and serves as an introduction to this volume of Springer Lecture Notes in Computer Science devoted to the "Conference on Business Process Management: On the Application of Formal Methods to Process-Aware Information Systems". The goal is to put the contributions to this conference into perspective. Section 1 puts BPM in its historical perspective going back to the late seventies. Section 2 defines BPM and compares it with workflow management. Based on this the paper zooms into the formal foundations of BPM on the one hand (Section 3) and technology and emerging standards for BPM on the other hand (Section 4). This way, the paper reflects the objective of this conference: Bringing together (computer) scientists and practitioners to work on advancing BPM methods, techniques, and tools.

References

1. W.M.P. van der Aalst. Three Good Reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Cambridge, Massachusetts, Nov 1996.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

3. W.M.P. van der Aalst. Making Work Flow: On the Application of Petri nets to Business Process Management. In J. Esparza and C. Lakos, editors, *Application and Theory of Petri Nets 2002*, volume 2360 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, Berlin, 2002.
4. W.M.P. van der Aalst and P.J.S. Berens. Beyond Workflow Management: Product-Driven Case Handling. In S. Ellis, T. Rodden, and I. Zigurs, editors, *International ACM SIGGROUP Conference on Supporting Group Work (GROUP 2001)*, pages 42–51. ACM Press, New York, 2001.
5. W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed. Pattern-Based Analysis of BPML (and WSCI). QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002.
6. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
7. W.M.P. van der Aalst and A.H.M. ter Hofstede. Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages. In K. Jensen, editor, *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, volume 560 of *DAIMI*, pages 1–20, Aarhus, Denmark, August 2002. University of Aarhus.
8. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. QUT Technical report, FIT-TR-2002-06, Queensland University of Technology, Brisbane, 2002.
9. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. QUT Technical report, FIT-TR-2002-02, Queensland University of Technology, Brisbane, 2002. (Also see http://www.tm.tue.nl/it/research/patterns.) To appear in *Distributed and Parallel Databases*.
10. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 2003 (to appear).
11. S. Burbeck. The Tao of e-Business Services. IBM Corporation, http://www-4.ibm.com/software/developer/library/ws-tao/index.html, 2000.
12. DAML-S and Related Technologies. www.daml.org/services/daml-s/0.7/survey.pdf, 2003.
13. C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, 1979. ACM Press.
14. C.A. Ellis and G. Nutt. Workflow: The Process Spectrum. In A. Sheth, editor, *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems*, pages 140–145, Athens, Georgia, May 1996.
15. H. Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, University of Twente, Enschede, The Netherlands, 2002.
16. Gartner. Gartner's Application Development and Maintenance Research Note M-16-8153, The BPA Market Cathes another Major Updraft. http://www.gartner.com, 2002.
17. A. W. Holt. Coordination Technology and Petri Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 278–296. Springer-Verlag, Berlin, 1985.
18. IDS Scheer. ARIS Process Performance Manager (ARIS PPM). http://www.ids-scheer.com, 2002.

19. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
20. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2003. Available via http://www.tm.tue.nl/it/research/patterns.
21. B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows (Revised version). QUT Technical report, FIT-TR-2002-03, Queensland University of Technology, Brisbane, 2002. (Also see http://www.tm.tue.nl/it/research/patterns.) To appear in *Acta Informatica*.
22. P. Lawrence, editor. *Workflow Handbook 1997, Workflow Management Coalition*. John Wiley and Sons, New York, 1997.
23. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
24. D.C. Marinescu. *Internet-Based Workflow Management: Towards a Semantic Web*, volume 40 of *Wiley Series on Parallel and Distributed Computing*. Wiley-Interscience, New York, 2002.
25. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
26. Dirk Wodtke and Gerhard Weikum. A formal foundation for distributed workflow execution based on state charts. In Foto N. Afrati and Phokion G. Kolaitis, editors, *Proceedings of the 6th International Conference on Database Theory – ICDT '97, Delphi, Greece, January 8–10, 1997*, volume 1186 of *Lecture Notes in Computer Science*, pages 230–246. Springer, 1997.
27. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Pattern-Based Analysis of BPEL4WS. QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.
28. M.D. Zisman. *Representation, Specification and Automation of Office Procedures*. PhD thesis, University of Pennsylvania, Warton School of Business, 1977.

Workflow: A Language for Composing Web Services

Giacomo Piccinelli[1] and Scott Lane Williams[2]

[1] University College London, Gower Street
WC1E 6BT London, United Kingdom
G.Piccinelli@cs.ucl.ac.uk

[2] Hewlett-Packard Software & Solutions, 11000 Wolf Road
95014-0691 Cupertino (CA), USA
Scott_L_Williams@HP.com

Abstract. The introduction of Web Services has a profound impact on component models. The interaction processes behind a service become integral part of the component type, and as such formally described and automatically manageable. Workflow emerges as the reference model for the description of interaction processes associated to individual web services. In the DySCo (Dynamic Service Composition) project, we investigate the use of workflow for both the modelling and implementation of composite solutions based on web services. Key aspect of DySCo is the separation between composition and coordination logic. In this paper, we discuss the composition model defined in DySCo, and a technology framework to enforce it.

1 Introduction

Since their early definition, composition has been a central concept for web services. From a business perspective, web services represent a new channel for the offer as well as the acquisition of business services. Peculiarity of the web service channel is that the interaction process between service provider and service consumer is completely automated. Beyond electronic data transfer, automation extends to all aspects of business interaction. Negotiation of service delivery terms and management of service-level agreements are just some examples. A comprehensive description of the foundational and operational aspects of the web service model can be found in [5,14].

In the web service model, the provider of a new web service $\mathcal{WS}$ drives the composition of internal and external capabilities in order to produce a new capability. Both internal and external capabilities are modelled as web services that act as service components for $\mathcal{WS}$. Similarly, $\mathcal{WS}$ can be used as service component for other web services. The fact that a capability is available internally or needs to be acquired externally reflects on the design of a service. Still, the web service model enforces a clear separation between a service component and the related service provider. Different providers can be used for the same capability under different circumstances, and the selection logic is integral part of the overall design of a web service.

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 13–24, 2003.

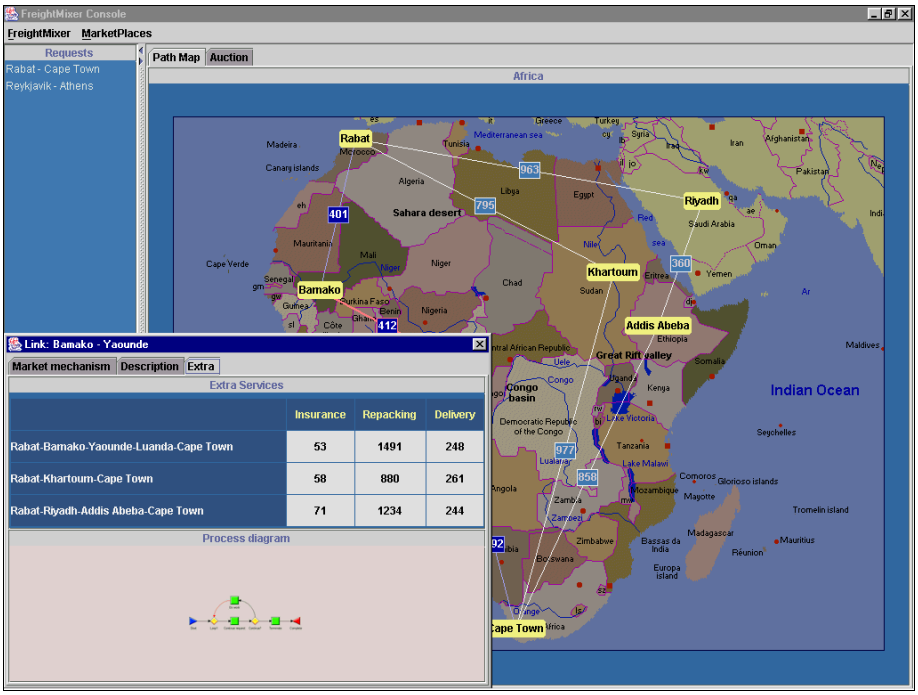

Fig. 1. Monitoring console in the prototype for the service composition system of FreightMixer

The decupling of web service and web service provider is closely related to the most noticeable feature of the web service model: process-oriented interfaces. In traditional component models, method signatures represent the only information on the interaction requirements of a component. While the invocation of a method can trigger complex interaction processes, interaction logic is not formally exposed. In the web service model, the interaction processes associated to a web service are explicitly formalized and exposed. WSCI (Web Service Choreography Language) [1] and WSFL (Web Service Flow Language) [8] are examples of languages for the formalization of interaction processes associated to web services. The rationale for process-oriented interfaces is composition in general, and dynamic composition in particular. The composition logic for a web service depends on the interaction processes of the service components for their orchestration. Interaction processes are central to the selection logic for service providers.

The objective of the DySCo (Dynamic Service Composition) [11] project is the development of a modelling and technology framework for the dynamic composition of web services. The distinctive element for DySCo with respect to other approaches [2,5,8,17] is the separation between composition logic and coordination logic. The designer can concentrate on the coordination of business capabilities, which are modelled as business roles. The coordination logic for the service providers is derived automatically based on the roles that each provider commits to play. In DySCo as

well as in the wider web service community, workflow [4,6] is the reference model for the formalization of both interaction logic and composition logic of web services.

In this paper, we first introduce (Section 2) a reference scenario for service composition based on the FreightMixer prototype. In (Section 3), we discuss some of the requirements for web service composition, with special focus on the dynamic aspects of composition. In (Section 4), we describe the composition model adopted in DySCo. In (Section 5), we present an overview of technology in the DySCo framework. In (Section 6), we report on related works. In (Section 7), we present some conclusions and remarks drawn from our involvement in DySCo.

2 Service Composition in FreightMixer

FreightMixer is a provider of freight services. Peculiarity of FreightMixer is a business model entirely based on web services. The company is not a real one, but it represents a realistic scenario for the use of web services. The scenario provides a conceptual framework that can be applied to a wide range of business contexts. More details can be found in [11]. In this section we summarize the service composition model used by FreightMixer.

The freight market is highly competitive, and customers explore multiple options before every shipment. Customers send RFQs (request for quote) to specific providers. Alternatively, customers can use auction-based instruments made available by electronic marketplaces. Switching costs are negligible, and the best offer wins the deal. Service providers are under pressure for competitive pricing of comparable levels of service. In terms of service execution, customer's main concern is that service-level agreements are respected (e.g. level of notification). The main concern for service providers is the cost of the execution infrastructure. Fixed costs are quite significant, and full utilization of resources is crucial. For example, the cost of shipping a container is almost independent from the content. Maximum usage of capacity translates into lower costs per unit.

FreightMixer offers an end-to-end freight service without owning fixed transport or transport-related infrastructure. When a new service request arrives, FreightMixer acquires services from other providers and combines them into a complete package. The result of the combined services constitutes the infrastructure upon which FreightMixer bases its offer. The composite nature of FreightMixer's offer is completely transparent to the customer. In terms of service execution, the providers for some of the service components may interact directly with the customer. Still, the interaction is on behalf and under the responsibility of FreightMixer. Providers of service components may also interact with each other. The interaction between service providers as well as with the customer depends on the overall composition logic decided by FreightMixer, and is regulated by the agreements between FreightMixer and each individual service provider.

In (Fig. 1) is represented the monitoring console for a prototype system that implements FreightMixer's operational model. The panel on the left contains the list of the service requests for which a solution is currently under development. Focusing on a specific service request (e.g. sending goods from Rabat to Cape Town), the main panel presents the various options under consideration. For each leg of each route, the

system presents the results of the ongoing negotiations. Expanding one of the legs, the system shows the ongoing negotiations for the specific leg. The system shows the various negotiation parameters, and in particular the service deliver process required to any service provider that wants to cover that leg. Service delivery processes are fundamental for service composition. Individual processes reflect the requirements for the provider of a leg of transport, the provider of the previous leg, the provider of the next leg, and the providers of corollary services such as insurance. Processes also reflect monitoring requirements from FreightMixer, and special requirements from the customer (e.g. notification). While not represented in the picture, the system simultaneously manages the negotiation with the customer. The system also monitors the execution of the complete solution if the deal is won. More details can be found in [11,13].

The core value of FreightMixer lays in the capability to aggregate the appropriate mix of service providers around a single objective. FreightMixer exploits competitive offer from other providers for the implementation of a solution matching the needs of its customers. The success of FreightMixer depends on the ability to acquire spare capacity at low cost, and to efficiently combine different resources. Service composition and negotiation are the pillars of FreightMixer's business model, and web services play a key role in their automation.

3 Requirements for Dynamic Composition

The concept of dynamic composition [10,16] is subject to different characterizations within the web service community. One aspect of web service composition is the binding between formal and actual provider for a service [14]. The set of service providers involved in an instance of a composite web service can be dynamically populated, hence the reference to dynamic composition. To avoid ambiguity, we refer to such aspect of composition as aggregation. Explicit management of aggregation logic is an important requirement for dynamic composition.

A second aspect of dynamic composition is the coordination of different web services [18]. The mainstream approach is to have a single entity responsible for coordination, hence the reference to coordination as orchestration. While agreeing on the need for a centre of responsibility in a composite service, our view on the orchestration model is that more can be added in order to capture the full potential for web service composition. In particular, we refer to peer-to-peer interaction and delegation. Using the FreightMixer example, the providers of two legs of transport route may need to interact with each other directly in the context of the overall service created by FreightMixer. We propose both delegation and peer-to-peer interaction as key requirements for dynamic composition.

Considering the application domain for web services, business applications represent the most important source of requirements for web services in general and composition in particular. The close relation between business services and web services has strong implications on web service solutions. Business processes are the driving force behind business services, and the association of business processes with workflow models and technology [4] has deep roots in the business world. Leveraging the conceptual framework and technology associated with workflow is a general

Table 1. Basic grammar for workflow processes. $\mathcal{N}$ is a set of variable names. $\mathcal{V}$ is a set of constant names (values). $\mathcal{R}$ is a set of resource names. $\mathcal{T}$ is a set of task names. The condition c in the *choice* is a binary function with domain $\mathcal{N}$.

$\mathbf{W ::= \varepsilon}$	*empty process*	
$\mid \mathbf{t_r(\sigma)}$	*task*	$(\sigma : \mathcal{N} \to \mathcal{V} \quad r \in \mathcal{R} \quad t \in \mathcal{T})$
$\mid \mathbf{W.W}$	*sequence*	
$\mid \mathbf{W +_c W}$	*choice*	
$\mid \mathbf{W \parallel W}$	*concurrency*	
$\mid \mathbf{! W}$	*loop*	

Table 2. Labelled Transition System (LTS) defining the process evolution function $\to$ in the algebra (W, $\to$)

$$\textbf{(step)}\ \ \Omega::\mathrm{tr}(\sigma) \xrightarrow{\varphi(t,\,r,\,\sigma')} \Omega \triangleleft \sigma' ::\varepsilon \qquad \text{where } \sigma' = \rho\,(t,\,r,\,\Omega \triangleright \sigma)$$

$$\textbf{(loop)}\ \ \Omega::!W \xrightarrow{\tau} \Omega::W.(!W)$$

$$\textbf{(seq1)}\ \ \frac{\Omega::W1 \xrightarrow{\alpha} \Omega'::W1'}{\Omega::W1.\ W2 \xrightarrow{\alpha} \Omega'::W1'\ .\ W2}$$

$$\textbf{(seq2)}\ \ \Omega::\varepsilon\ .\ W2 \xrightarrow{\tau} \Omega::W2$$

$$\textbf{(choice)}\ \ \Omega::W1 +_c W2 \xrightarrow{\tau} \Omega::WX \quad \textit{where } X = \mathrm{eval}(c) \in \{1,2\}$$

$$\textbf{(comp1)}\ \ \frac{\Omega::W1 \xrightarrow{\alpha} \Omega'::W1'}{\Omega::W1 \parallel W2 \xrightarrow{\alpha} \Omega'::W1' \parallel W2}$$

$$\textbf{(comp2)}\ \ \frac{\Omega::W2 \xrightarrow{\alpha} \Omega'::W2'}{\Omega::W1 \parallel W2 \xrightarrow{\alpha} \Omega'::W1 \parallel W2'}$$

$$\textbf{(comp3)}\ \ \frac{\Omega::W1 \xrightarrow{\alpha} \Omega'::W1' \qquad \Omega::W2 \xrightarrow{\beta} \Omega''::W2'}{\Omega::W1 \parallel W2 \xrightarrow{\mu(\alpha,\beta)} \Omega' \bigtriangledown \Omega'' ::W1' \parallel W2'}$$

requirement for web services. In the specific case of composition, workflow represents an important source of opportunities as well as requirements. Coordination of different capabilities towards the achievement of a specific objective is a fundamental concept in the workflow model. A mapping between workflow and web services can immediately leverage existing expertise on the business side as well as on the technical side. On the technical side, integration between web service and workflow technology can provide an important integration route towards existing business applications.

4 Web Service Composition in DySCo

The objective for DySCo can be summarized as a composition framework for web services that match the requirements of a company such as FreightMixer. Initial results on the composition methodology developed in DySCo can be found in [11]. The aggregation aspects of the methodology are described in [13]. In this paper, we discuss the composition model defined in DySCo. The focus is on the separation enforced by the model between composition and coordination logic.
The composition model defined in DySCo is based on the workflow model. Both composition and coordination logics are specified as workflows. The designer of the web service defines the composition logic once. Specific tools in the DySCo framework automatically derive the coordination logic required for different aggregation patterns. Delegation and peer-to-peer interaction are integral part of both the composition and the coordination models.
In the remaining part of this section, we first give an overview of the workflow model (Section 4.1). We then describe in more detail the composition (Section 4.2) and coordination (Section 4.3) models defined in the DySCo framework.

4.1 The Workflow Model

The objective of workflow [6] is to model and manage the coordination of a set of resources towards the achievement of a given result. Resources are modelled as independent and self-contained units that have the capability to perform specific tasks. While resources may be capable of autonomous behaviour, the tasks specified in a workflow are performed only upon request. The coordination logic in the workflow model is based on an orchestration approach. A single logical entity is in charge of maintaining the state of the process, and to request the resources to perform tasks. An algebraic characterisation $(W, \rightarrow)$ for workflow processes is given in (Tab. 1 and Tab. 2).

The atomic element in the specification of a workflow process (Tab. 1) is the task. A task is defined by a name that indicates the activity required to a resource, as well as a name that indicates the resource that has to perform the activity. The specification for a task includes indications on the information that is supplied to a resource, and on the information that the resource will produce as a result of the activity. The flow logic for the process is expressed (Tab. 1) by the sequence, choice, concurrent, and loop operators. An operational definition for the execution semantics of a workflow process is described by the LTS in (Tab. 2).

The execution semantics for a workflow process is based on the concept of a global state Ω that contains a selection of the information generated during the execution of the process. The execution of a task (Tab. 2 - step) relies on a function ($\triangleright$) to extract form Ω the information required by the resource. In the same way, a function ($\triangleleft$) feeds into Ω new information generated by the resource. Only one resource is involved in each task ($r \in \mathcal{R}$). Transactional access to Ω is not explicit in the basic workflow model. The execution semantics also defines a concept of visibility over the evolution of the process. In the case of a task, a function φ defines the information to

make visible based on the task name, the resource name, and the information involved in the execution of the task itself.

A detailed discussion of the theoretical framework for workflow is outside the scope of this paper, and more information can be found in [7,9]. The reason why we focus our description on tasks more than on flow control will become apparent in the next section.

4.2 The DySCo Composition Model

One of the main limitations that prevent the traditional workflow model from fully exploiting the compositional potential of web service is the focus on activities. The unit of process in a traditional workflow revolves around asking a resource to perform an activity (task). The result from one task then constitutes the input for other tasks. The resource involved in a task is not aware of the resources involved in other tasks. The interaction between different resources is mediated by the process manager, which acts as logical point of concentration for the data flow associated to the process.

The composition model we propose for web services retains the overall structure of traditional workflow (Tab. 1 and Tab. 2), with exception of the task. In DySCo the focus shifts from activity to interaction. Resources become roles, and the task becomes an interaction step. In an interaction step, sets of roles interact towards the achievement of a given objective. In traditional workflow, the task name contains sufficient information for the resource to understand precisely the activity to perform. In an interactive step, the step name contains sufficient information for the roles involved to understand precisely the way in which they have to interact with each other. As in traditional workflow, information from the overall state of the process may be provided as input for the execution of an interactive step. An interactive step may produce output information that contributes to the state of the overall process.

Given the set $\mathcal{R}$ of role names and the set $\mathcal{S}$ of names of interactive steps, the entry for the task in the process grammar of (Tab.1) becomes:

$$s_r(\sigma) \quad \textit{interactive step} \quad (\sigma : \mathcal{N} \rightarrow \mathcal{V} \quad r \subseteq \mathcal{R} \quad s \in \mathcal{S})$$

The entry (step) for the process evolution function in the LTS of (Tab. 2) is replaced by:

$$\textbf{(int-step)} \quad \Omega :: s_r(\sigma) \xrightarrow{(t, v, \sigma')} \Omega \triangleleft \sigma' :: \varepsilon \quad \text{where} \quad \sigma' = \rho(s, r, \Omega \triangleright \sigma) \quad r \subseteq \mathcal{R} \quad v \subseteq \mathcal{R}$$

The involvement of multiple roles in an interactive step affects both the output function ρ and the visibility function φ for the step. The output function takes into consideration the contributions from all the roles involved in the step. The visibility function can give different perspectives on the execution of the interactive step depending on the set of roles defined by the observer. The granularity as well as the complexity of the interaction can vary between steps, but interactive steps maintain the atomicity property typical of tasks in traditional workflow.

The use of roles enforces a level of indirection between capabilities and capability providers. In particular, the interaction between the multiple roles in an interaction step is independent from the providers that will cover each role. Similarly to resource names in traditional workflow, role names are used consistently across different steps in one process. The equivalent of a task in traditional workflow can be obtained with an interactive step that specifies only one role. Similar conditions can also be created in the aggregation phase if one provider covers all the roles in an interactive step.
Interactive steps and direct interaction between roles enable explicit modelling of peer-to-peer interaction, as well as delegation. Peer-to-peer interaction and delegation constitute the basis for both the coordination and the aggregation models adopted in DySCo.

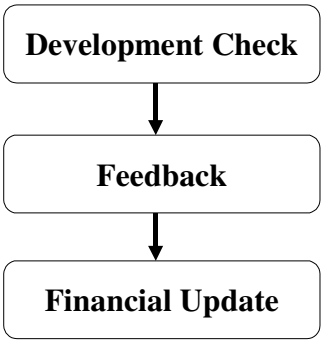

Fig. 2. Section of process modelled by FreightMixer. The customer first verifies the stage of delivery. There is then an exchange of information between customer and transport provider. The final step involves the interaction between customer and a financial institution.

4.3 The DySCo Coordination Model

The coordination model adopted in DySCo operates at role level, and is entirely based on peer-to-peer interaction. The approach is to derive from a DySCo process D a set of traditional workflow processes $\{P_j\}$, such that the result of the combination of all the P_j is equivalent to D. A detailed description of the equivalence relation is outside the scope of the paper, but the general idea is that it is based on the classic notion of bisimulation [9]. The visible aspects of process execution are modelled by functions such as φ (Tab.1) and λ (int-step).
Given a DySCo process D and the set $\mathcal{R}$ of all the roles involved in D, let us consider $\mathcal{P}(\mathcal{R})$ the set of all the possible subsets of $\mathcal{R}$. Let us also consider $C(\mathcal{P}(\mathcal{R}))$ the set of all the subsets of $\mathcal{P}(\mathcal{R})$, such that $c \in C(\mathcal{P}(\mathcal{R}))$ implies that the union of all the sets in c is equal to $\mathcal{R}$. Given a set $c = \{c_j\} \in C(\mathcal{P}(\mathcal{R}))$ the coordination model defines a set $W = \{W_j\}$ of traditional workflow processes such that there is a one-to-one relation between the elements of c and W. Each process W_j contains the orchestration logic for the web services of the provider covering the roles in the corresponding c_j, as well as

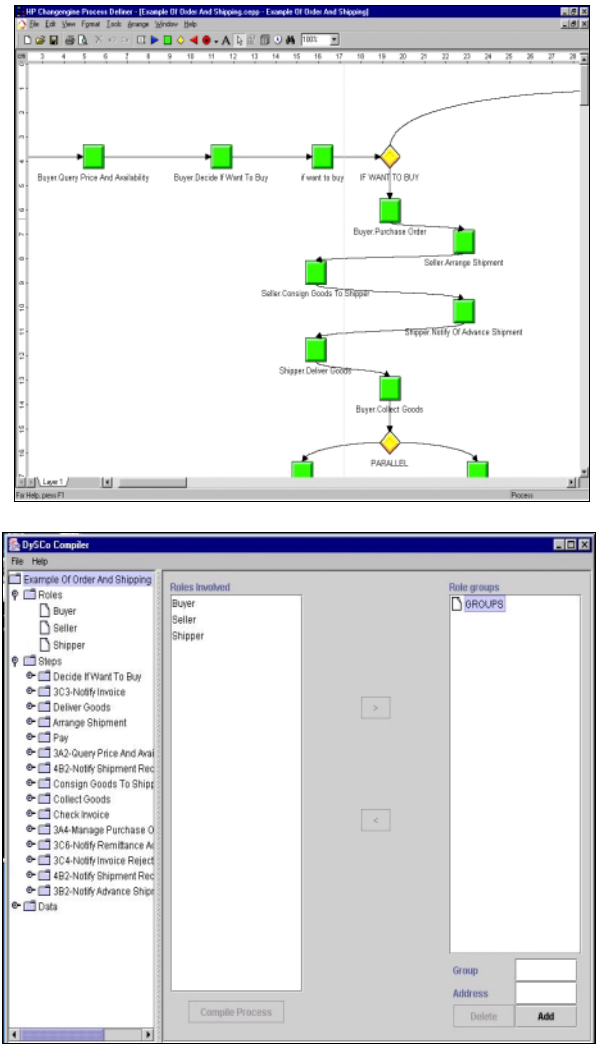

Fig. 3. Two components of the DySCo implementation framework. On the left, the design environment for DySCo processes. On the right, the projection generator (view on the creation of role groups).

the interaction and synchronisation logic with other providers. The interaction logic in W_j refers to the exchange of service-level information between providers. Synchronisation logic refers to the signalling between providers required in order to align the global flow of execution.

For example, FreightMixer may have a DySCo process F (Fig. 2) related to a service offer based on frequent customer updates. One of the steps in the process can be *Development Check,* which specifies the gathering of information by some roles (e.g. *customer*) from other roles (e.g. *transport provider* and *financial institution*). For the entity covering the role *transport provider*, the W_t related to F describes the usage logic for internal service capabilities (e.g. to generate a report). W_t also indicates to

send the report to *customer*, or to expect *customer* to potentially ask for it. Copy of the report is sent also to the *financial institution*. The W_c for the *customer* specifies to wait for a communication from the *transport provider*, or the option to ask for the report. The next step in F is *Feedback*, which specifies an exchange of validation messages between some roles (e.g. *customer*) and other roles (e.g. only to the *transport provider*). W_c and W_t capture the activity required to the related entities. W_c, W_t, and W_f (financial institution) also include synchronisation mechanisms such that the financial institution does not start the activities involved in *Financial Update* until *Feedback* is completed.

The coordination model preserves the level of indirection introduced by the composition model between a capability (captured by a role) and the web services that implement the capability. Such indirection has direct consequences on delegation as well as aggregation. In terms of delegation, the main implication is that trust models for service providers can be based on the roles they play in the composite solution, in addition to the web services they offer. In terms of aggregation, the shift is from the provision of sets of web services to the fulfilment of a specific role. The impact on models and techniques for negotiation are described in [13].

5 The DySCo Implementation Framework

The DySCo implementation framework includes development tools (Fig. 3) as well as execution infrastructure to support the complete lifecycle of a composite web service. In this section, we describe the part of the framework related to the definition of the composition logic for the web service. We also describe the components involved in the automatic generation of the coordination logic for specific configurations of role distributions.

The composition logic for a web service is described using a graphical environment (Fig. 3, left) based on an extension of the commercial tool iGrafx™. The process model derives from the reference standard for workflow processes specified by the WfMC (Workflow Management Coalition) [6]. The design environment for DySCo covers processes as well as steps. In addition to the flow operators in the basic workflow model (Tab. 1), specific design patterns allow the use of higher-level constructs. Multiple branching and conditional loops are examples of such constructs. The use of sub processes is also possible. Concerning interactive steps, the interaction logic for the roles is based on an abstract execution environment also defined in DySCo. The abstract execution environment includes concepts such as a virtual repository shared by the roles, as well as direct role-to-role interaction (see [11] for more detail). The specification of the steps is based on workflow. The design environment is the same for both processes and steps. Libraries can be created for processes as well as steps. As an example of library for interactive steps, we captured and collected in one library all the currently specified PIPs (Partner Interaction Processes) specified in the RosettaNet standard [12,15].

The composition logic for the web service involves multiple roles. The coordination logic for the actual providers (and consumers) involved in the different instances of the web service depends on the roles that each party covers. The projection generation environment (Fig. 3, right) allows the automatic generation of the coordination logic for different configurations of role groups. When the files

containing the composition logic are loaded into the projection generator, the tool automatically derives all the roles involved. It is then possible to specify a number of group names, and the association between roles and groups. Once all the roles have been assigned to at least one group, the tool generates for each group the workflow processes describing the related coordination logic. The activity can be repeated for different role groups. As specified in the DySCo coordination model, the projection generator adds synchronisation operations to the service logic required to the roles in a group. The control over the algorithms used for the creation of the projections makes possible formal proofs of equivalence between projections and the process from which they derive.

6 Related Works

Resulting from the convergence of XLANG [17] and WSFL [8], BPEL4WS [2] is the main representative of a recent stream of activities promoting workflow as the reference model for the composition of web services. The focus of BPEL4WS is on the internal aspects of a composite solution. WSCI [1] (led by SUN) is the main representative of the complementary stream of activities promoting workflow as a model for the external aspects of web services. In terms of technology platforms, almost every commercial product for workflow management currently supports or has plans to support web services.

Currently focused on the more general problem of business-to-business (B2B) integration, the work of organisations such as ebXML [3] and RosettaNet [15] plays a fundamental role for web services. The creation of a common ontology including dictionaries as well as processes is a prerequisite for an effective automation of composition. The W3C organisation [5] acts as aggregation point for the work on architectures, protocols, and description models for web services.

7 Conclusions

While composition in general is at the centre of the web service model, the application context for web services sets specific requirements on composition models. Business applications and business services represent the main application domain for web services, and workflow represents the most established conceptual framework for the composition of business capabilities. We propose that workflow can play a central role for web services, both in terms of model and technology.

In the DySCo (Dynamic Service Composition) project, the workflow model becomes the basis for a multi-layered composition framework for web services. The distinctive aspect of DySCo is the separation of composition logic from coordination and aggregation logic. The model enforces a level of indirection between the capabilities involved in a composite solution and the configuration of web services that deliver such capabilities. The implementation framework for DySCo makes explicit reuse of workflow technology for the modelling and enactment of composite solutions.

References

1. Arkin A., and Alt.: Web Services Choreography Interface (WSCI) 1.0. W3C Note (2002)
2. Curbera F., and Alt.: Business Process Execution Language for Web Services (BPEL4WS) 1.0. IBM online resources, (2002)
3. ebXML: Reference Web Site for the Organization. http://www.ebXML.org (2002)
4. Georgakopoulos D., Hornick M.F., Sheth A.P.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. In: Distributed and Parallel Databases Vol. 3 No. 2 (1995)
5. Haas H., and Alt.: Web Services Activity. W3C Working Groups (2002)
6. Holligsworth, D.: The Workflow Reference Model. Workflow Management Coalition (WfMC) (1994)
7. Hoare C.A.R.: Communicating Sequential Processes. In: Communication of the ACM, Vol. 21 No. 8 (1978)
8. Leymann, F.: Web Services Flow Language (WSFL 1.0). IBM (2002)
9. Milner R.: Communication and Concurrency. Prentice-Hall (1989)
10. Nierstrasz, O., and Meijler, T. D.: Requirements for a Composition Language. In: Ciancarini, P., Nierstrasz, O., Yonezawa, A. (Eds.): Object-Based Models and Languages for Concurrent Systems, LNCS 924 (1995)
11. Piccinelli G., Di Vitantonio G., and Mokrushin L.: Dynamic Service Aggregation in Electronic Marketplaces. In: Computer Networks Journal, Vol. 37 No. 2, Elsevier Science (2001)
12. Piccinelli G., Finkelstein A., Stammers E.: Automated Engineering of e-Business Processes: the RosettaNet Case Study. In: Proc. 6th Int. Conf. on Systemic, Cybernetics, and Informatics, Orlando, Florida, USA (2002)
13. Piccinelli G., Preist C., and Bartolini C.: E-Service Composition: Supporting Dynamic Definition of Process-oriented Negotiation Parameters. In: Proc. IEEE 2nd e-Negotiations Workshop, Munich, Germany (2001)
14. Kuno H.: Surveying the E-Services Technical Landscape. In: Proc. 2nd Int. Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS). Milpitas, California, USA (2000)
15. RosettaNet: Reference Web Site for the Organization. http://www.RosettaNet.org (2002)
16. Seaborne A., Stammers E., Casati F., Piccinelli G., and Shan M.: A framework for business composition. In: Proc. W3C Workshop on Web Services, San Jose, CA, USA (2001)
17. Thatte S.: XLANG – Web Services for Business Process Design. Microsoft (2001)
18. Stearns M. and Piccinelli G.: Managing Interaction Concerns in Web-Service Systems. In: Proc. 2nd IEEE Int. Workshop on Aspect Oriented Programming for Distributed Computing Systems (AOPDCS), Vienna, Austria (2002)

Mining Most Specific Workflow Models from Event-Based Data

Guido Schimm

OFFIS, Escherweg 2,
26121 Oldenburg, Germany
schimm@offis.de

Abstract. This paper presents an approach on mining most specific workflow models from event-based data. The approach is embedded in the context of data mining and knowledge discovery in databases. It consists of two parts. The first one is an introduction of a block-structured workflow model representation and the second one is an extraction procedure for workflow models based on that model representation. This paper describes both parts in detail and also outlines preceding and subsequent steps.

1 Introduction

Today, workflow management systems are applied in many organizations. Setting up and maintaining a workflow management system require workflow models which prescribe how business processes should be managed by the system. Typically, the user is required to provide these models. Constructing process models from scratch is a difficult and error-prone task that often requires the use of an expert. An alternative way to construct workflow models from scratch is to extract them from event-based data captured in form of logs from running business processes. The goal of workflow mining is to extract workflow models for business processes from such logs.

In this paper we present an approach on mining most specific workflow models. A model is considered to be most specific according a given log if it is complete and minimal. Completeness describes that a model should preserve all the dependencies between activities that are present in the log. Minimality assures that the model should neither introduce paths of execution nor spurious dependencies between activities which are not present in the log.

Let us assume that the execution of process instances from which we have captured a log is controlled by knowledge that only the actors have in mind. This process-related knowledge is called the implicit model. In case that a log does not cover all possible ways of executing a process, the mined workflow model may be different from the implicit model due to the fact that it contains dependencies between activities which are not part of the implicit model. The obvious limitation of mining workflow models is that the quality of a mined model, with respect to its implicit model, depends on how much the log covers

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 25–40, 2003.

the implicit model. Also, its characteristic to be most specific can only be related to the log, not to the implicit model itself.

The rest of the paper is organized as follows. The following section outlines the context in which our approach is embedded. In section 3 we define the input. A workflow model representation is outlined in section 4. In section 5 we describe our mining process in detail, followed by a short description of engineering the output in section 6. An experimental evaluation is outlined in section 7. Subsequently, we discuss related work and conclude the paper with a summary.

2 Mining Framework

We consider the mining of workflow models from event based data to be a process of *knowledge discovery in databases* (KDD)[1]. A KDD process roughly consists of the following steps: data consolidation, selection and preprocessing, data mining, interpretation and evaluation. In this paper, we focus on the data mining step and its interfaces to the preceding and subsequent steps.

Generally, data mining consists of two tasks. The first task is to define a model representation. Based on this the second task is the extraction of models from data using appropriate algorithms. Workflow mining can be considered a data mining method that uses a particular workflow meta-model as model representation and specialized algorithms for extracting workflow models from logs. The interfaces of the workflow mining step to its preceding and subsequent steps are given by a definition of its input and a transformation of its output.

3 Defining the Input

Monitoring process instances provides a large amount of data of different events that occur while an instance is being executed. Because we are interested in extracting models that describe control flows between activities within processes we consider events which are related to the life cycle of an activity.

Let us use the finite state machine depicted in Figure 1 to describe the life cycle of an activity. In this paper, we focus on two kinds of events: *Start* and *Complete*. An event of kind *Start* marks the transition from the state *Scheduled* into the the state *Active*. An event of kind *Complete* marks the transition from the state *Running* into the state *Completed*.

For each activity captured in form of a pair of *Start* and *Complete* events we distinguish its type, occurrence, and instance. Two activities of the same type, i.e. the same kind of activity, are differentiated into multiple occurrences if they are embedded in different contexts. The context of an activity is defined by the set of other activities for which a precedence relation to that activity exists. Often, it is enforced that each activity within the same process is named differently. In this case we capture exactly one occurrence per activity. We capture an activity occurrence that is executed at least ones, i.e. we get one activity instance. We

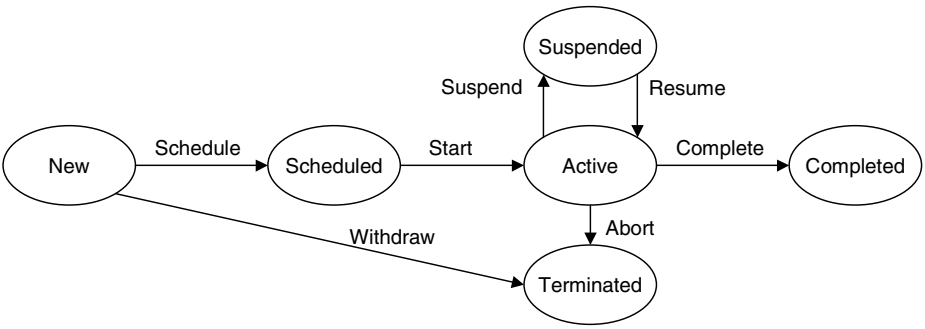

Fig. 1. Finite state machine for the life cycle of an activity

read multiple instances of an activity occurrence if it is performed repeatedly, e.g. inside a loop.

We log an instance of a business process executed under the control of a monitoring tool in form of a trace. A trace consists of a set of events of the life cycle of each activity performed inside a particular process instance. It is defined as follows.

Definition 1. *Let* $e = (c, o, i, s, t)$ *denote an event of starting or completing the instance* $i (i \in \mathbf{N})$ *of an activity occurrence* o *at the logical point in time* $t (t \geq 1)$ *as part of executing a process instance* c*, where* $s \in \{Start, Complete\}$ *is called stereotype of* e*. A trace* $E(c) = (\{e = (c', o, i, s, t) \mid c' = c\}, <_t)$ *is the set of all events of an instance* c *ordered by time, where every event has a unique* t*.*

Mining a model for a business process is based on a set of traces, called the log of the business process. A log is defined as follows.

Definition 2. *Let* $C(p) = \{c_1, \ldots, c_n\}$ *be a set of instances for a particular business process* p*. A log* $L(p) = \{E(c) \mid c \in C(p)\}$ *is a set of traces captured for* p*.*

For a particular business process we select the traces into a log. Before we can start mining we check the log in order to detect inconsistencies. We expect each activity instance in each trace to have exactly one event of stereotype *Start* and one event of stereotype *Complete*, such that each *Start* event comes always before the corresponding *Complete* event. Inconsistent traces are eliminated.

4 Model Representation

The model representation, i.e. the meta-model the extracted workflow models are based on, is a block-oriented model. It defines that each workflow model consists of an arbitrary number of nested building blocks. Building blocks are differentiated into operators and terms. Operators combine building blocks and

define the control flow of a workflow model. Basic terms are references for activity occurrences or sub-workflows that are embedded into a control flow. We use references to activity occurrences instead of activity occurrences because this allows us to have partial synchronization for an activity occurrence.

We build a block-structured model top-down by setting one operator as starting point of the workflow model and nest other operators until we get the desired control flow structure. At the bottom of this structure we embed basic terms into operators and terminate the nesting process. Therefore, each process model can be represented in form of a tree whose leafs are always operators. Beside the tree representation of block-structured models we can write them in form of terms. Furthermore, block-structured models can be represented in form of diagrams, too. Block-structured models have some advantages: They are well-formed and always sound. Therefore, using this kind of meta-model we are sure that the extracted workflow models do not contain any deadlocks or other anomalies.

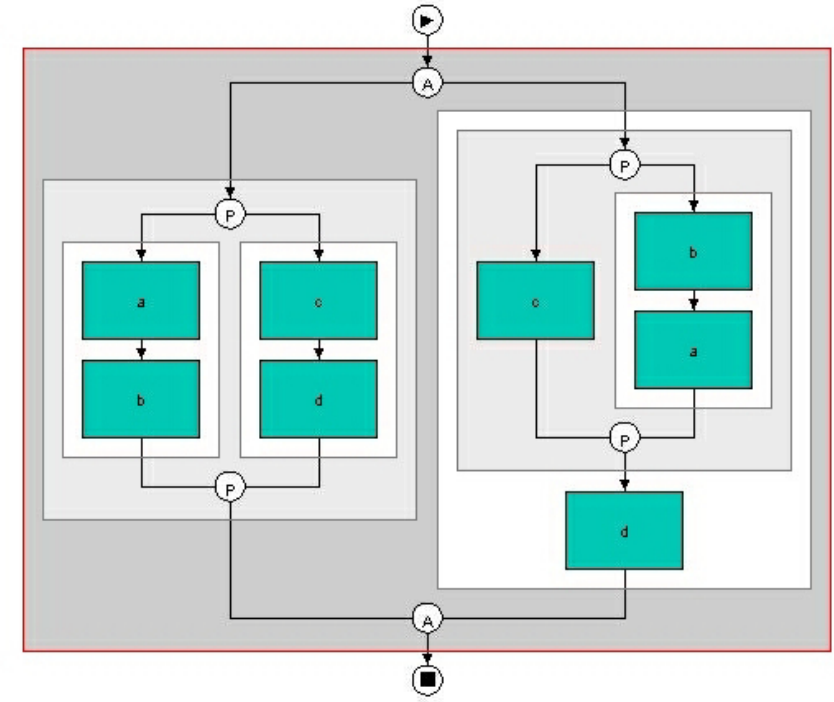

Fig. 2. A simple process model

In this paper we use a basic block-oriented meta-model that consists of activity occurrence references as basic terms and n-ary operators for sequences, parallels, and alternatives. The *Sequence* operator defines that all embedded blocks are performed in a sequential order. Let $\mathcal{S}$ denote the *Sequence* operator

and let a and b denote two activity occurrence references, then $\mathcal{S}(a, b)$ is a workflow model defining that activity a is performed before activity b. In contrast, the *Parallel* operator defines that all embedded blocks can be executed in parallel, i.e. that no precedence relation exists between the embedded blocks. Let $\mathcal{P}$ denote the *Parallel* operator, and let a and b denote two activity occurrence references, then $\mathcal{P}(a, b)$ is a workflow model defining that activity a and activity b can be performed independently, i.e. in any order or in parallel. The *Alternative* operator defines a choice of exactly one block out of all its embedded blocks. This operator is supplemented by a set of rules determining the choice. Let $\mathcal{A}$ denote the *Alternative* operator, and let a and b denote two activity occurrence references, then $\mathcal{A}(a, b)$ is a workflow model defining that either activity a or activity b is executed. Additionally, we define a *Loop* operator $\mathcal{L}$. The *Loop* operator contains only one block that is executed repeatedly until its loop condition holds.

For example, Figure 2 shows the process model $\mathcal{A}(\mathcal{P}(\mathcal{S}(a, b), (c, d)), \mathcal{S}(\mathcal{P}(c, \mathcal{S}(b, a)), d))$ in form of a diagram. The root of this model is a alternative operator that contains all other blocks in a hierarchical manner, such that all activity occurrence references are embedded in a nested control flow.

For our meta-model we define an algebra that consists of a set of axioms covering distributivity, associativity, commutativity, and idempotency. These axioms are the basis of term rewriting systems that can be used in order to transform workflow models. A set of basic axioms used by our workflow mining approach is shown in Table 1. Associativity is defined for all operators (A2, A5, A8). Distributivity is defined in form of left distributivity of the *Sequence* operator over the *Alternative* operator (A4), and in form of full distributivity of the *Parallel* operator over the *Alternative* operator (A7). Note that this algebra does not contain an axiom for the right or full distributivity of the *Sequence* operator over the *Alternative* operator because this would lead to different models depending on the time a decision is made. Commutativity of embedded blocks is defined for *Alternative* and *Parallel* operators (A1, A6). Idempotency is only defined for the *Alternative* operator (A3).

Table 1. Basic algebra

A1: $\mathcal{A}_1(x_1, \ldots, x_n) = \cdots = \mathcal{A}_{n!}(x_n, \ldots, x_1)$
A2: $\mathcal{A}(\ldots, x_1, \ldots, x_n, \ldots) = \mathcal{A}(\ldots, A(x_1, \ldots, x_n), \ldots)$
A3: $\mathcal{A}(x, x, \ldots) = x$
A4: $\mathcal{S}(\mathcal{A}(x_1, \ldots, x_n), y_1, \ldots, y_m) =$
$\quad \mathcal{A}(\mathcal{S}_1(x_1, y_1, \ldots, y_m), \ldots, \mathcal{S}_n(x_n, y_1, \ldots, y_n))$
A5: $\mathcal{S}(\ldots, x_1, \ldots, x_n, \ldots) = \mathcal{S}(\ldots, \mathcal{S}(x_1, \ldots, x_n), \ldots)$
A6: $\mathcal{P}_1(x_1, \ldots, x_n) = \cdots = \mathcal{P}_{n!}(x_n, \ldots, x_1)$
A7: $\mathcal{P}(\mathcal{A}(x_1, \ldots, x_n), y_1, \ldots, y_m) =$
$\quad \mathcal{A}(\mathcal{P}_1(x_1, y_1, \ldots, y_m), \ldots, \mathcal{P}_n(x_n, y_1, \ldots, y_n))$
A8: $\mathcal{P}(\ldots, x_1, \ldots, x_n, \ldots) = \mathcal{P}(\ldots, P(x_1, \ldots, x_n), \ldots)$

The basic meta-model can be supplemented with further operators and basic terms. For example, one can define a *Parallel* operator that controls the order of starting its embedded blocks as well as a non-exclusive *Alternative* operator. Also, an basic term that represents sub-processes may be defined. In this paper we use only the basic meta-model outlined above.

5 Mining Workflow Models

In the previous sections we have defined the input of the mining process and a model representation. Now, we can describe how to mine most specific workflow models from input. Our mining process consists of five steps which are performed on a log $L(p)$ for a process p in sequential order. Following, each step is described in the order of application.

5.1 Labeling Multiple Activity Instances

A trace may contain events for multiple instances of the same activity occurrence. These instances result from executing an activity occurrence repeatedly within a process instance. Due to the fact that a workflow model only contains references for activity occurrences, we now substitute each instance of a set of multiple instances by a single instance of an activity occurrence. For this purpose we define the following labeling operator.

Definition 3. *Let $E(c)$ denote a trace, $E(c) \in L(p)$. The operator $\zeta : E \to E$,*

$$\zeta(e) = \begin{cases} e : & |\{e' \mid e, e' \in E(c) \wedge e'.o = e.o \wedge e'.s = e.s\}| = 1 \\ (c, k, 1, s, t) : & \text{otherwise, where } k = o \circ i \text{ represents a concatination} \end{cases}$$

is called labeling operator.

The labeling operator is applied to each event in each trace of the log $L(p)$. It transforms events, such that each instance of a set of multiple instances of an activity occurrence within a particular $E(c)$ is represented by a single instance of an activity occurrence that is artificially differentiated by the label $k = o \circ i$.

5.2 Grouping Traces into Classes

In this step, we group traces of a log into trace classes. The way of grouping depends on the meta-model. For the basic meta-model the building of trace classes is defined as follows.

Let us consider a trace. Its sequence of events can be splitted into alternating subsequences which only contain events of the same stereotype. Each subsequence is called a cluster of the trace.

Definition 4. *Let* $\gamma(E(c)) = \{E(c)_1^+, E(c)_1^-, E(c)_2^+, \ldots, E(c)_q^-\}$ *denote the clustering of* $E(c)$*, where* $E(c)_1^+$ *is the first set of subsequent events of stereotype* Start*,* $E(c)_1^-$ *is the first set of subsequent events of stereotype* Complete*, and so on.*

The type of an event is a tuple containing the activity occurrence and the stereotype. For the basic meta-model we define the equivalence of traces as follows.

Definition 5. *Let* $\kappa : \{(C, O, \mathbf{N}, S, T)\} \rightarrow \{(O, S)\}$*, with* $\kappa(e) = (e.o, e.s)$*, denote a projection on event* e*, where* o *denotes the activity occurrence of* e *and* s *denotes its stereotype.* $\kappa(e)$ *is called the type of* e*. Two events* e *and* e' *are equivalent,* $e \equiv e'$*, iff they have the same type,* $\kappa(e) = \kappa(e')$*. Two traces* $E(c)$ *and* $E(c')$ *are equivalent,* $E(c) \equiv E(c')$*, iff clusters at the same position contain the same subset of equivalent events,* $\forall E(c)_i^{+/-} \in \gamma(E(c)), \forall E(c')_i^{+/-} \in \gamma(E(c')) : \{\kappa(e) \mid e \in E(c)_i^{+/-}\} = \{\kappa(e') \mid e' \in E(c')_i^{+/-}\} \Leftrightarrow E(c) \equiv E(c')$*.*

Note, that the order of events within a cluster is not relevant. A trace class can now be defined as an ordered set of clusters that contains the event types of its member traces.

Definition 6. *Let* $\rho(p)_h$ *denote a part* $\{E(c_1), \ldots E(c_m) | E(c_1) \equiv \cdots \equiv E(c_m)\}$ *of equivalent traces of a partition of* $L(p)$ *based on the equivalence relation* $E(c) \equiv E(c')$*.* $D_h = \{U_1^+ = \{\kappa(e) \mid e \in \cup E(c_i)_1^+\}, U_1^- = \{\kappa(e) \mid e \in \cup E(c_i)_1^-\}, \ldots, U_q^- = \{\kappa(e) \mid e \in \cup E(c_i)_q^-\} \mid \forall E(c_i) : E(c_i) \in \rho(p)_h, 1 \leq i \leq |\rho(p)|\}$ *is the trace class of* $\rho(p)$*, where* $U_l^{+/-}, 1 \leq l \leq q$*, denote the clusters of event types.*

For a log $L(p)$ we can now determine the partition $D(L(p))$ whose parts are trace classes $(D_h)_{1 \leq h \leq l}$ consisting of equivalent traces. The number of trace classes for $D(L(p))$ depends on the variability of the process instances captured in $L(p)$. It ranges from 1 to the number of traces.

5.3 Extracting Precedence Relations

For each trace class D_h, $D_h \in D(L(p))$, we now extract a precedence relation. It relates two event types for which we detect a dependency.

Algorithm 1. *Let* $R_h = \{(o, o') \mid o \in O \cup \alpha, o' \in O\}$ *denote the precedence relation for* D_h*, where* α *is an initiating activity occurrence that marks the start of a process. Let* $R(L(p)) = (R_h)_{1 \leq h \leq k}$ *denote the family of all precedence relations for* $D(L(p))$*. Let* $\mu : U \rightarrow \{o \mid (o, s) \in U\}$ *be a projection function that returns all activity occurrences of the event types of a cluster* U*. Let* $\varphi : U \rightarrow \{Start, Complete\}$ *denote a function that returns the stereotype of a cluster* U*.* S *is a set. Input:* $D(L(p))$*.*

For each $D_h, D_h \in D(L(p))$ {
  $R_h := \emptyset$
  $S := \alpha$
    For each $U_i^{+/-}, U_i^{+/-} \in D_h, i = 1, \ldots, q$ {
      If $\varphi(U_i^{+/-}) = Start$ {
        $R_h := R_h \cup \{(o, o') | o \in S, o' \in \mu(U_i^+)\}$
      }
      Else { $S := S \cup \mu(U_i^-)$ }
    }
  }
  $R(L(p)) = R(L(p)) \cup R_h$
}
Return $R(L(p))$

Algorithm 1 shows that we use transitive dependencies between activity occurrences, e.g. $\forall o, o', o'' \in \bigcup \mu(U_i), U_i \in D_h : (o, o') \in R_h \wedge (o', o'') \in R_h \Rightarrow (o, o'') \in R_h$. Also note that we extract the precedence relation for each trace class separately.

Example 1. Let $D_1 = \{\{(a, Start), (c, Start)\}, \{(a, Complete)\}, \{(b, Start)\}, \{(c, Complete)\}, \{(d, Start)\}, \{(b, Complete), (d, Complete)\}\}$, $D_2 = \{\{(a, Start), (c, Start)\}, \{(c, Complete)\}, \{(d, Start)\}, \{(a, Complete)\}, \{(b, Start)\}, \{(b, Complete), (d, Complete)\}\}$, and $D_3 = \{\{(c, Start), (b, Start)\}, \{(c, Complete), (b, Complete)\}, \{(a, Start)\}, \{(a, Complete)\}, \{(d, Start)\}, \{(d, Complete)\}\}$ be the trace classes for a log. Figure 3 shows the precedence relations R_i that is extracted from D_i by applying Algorithm 1.

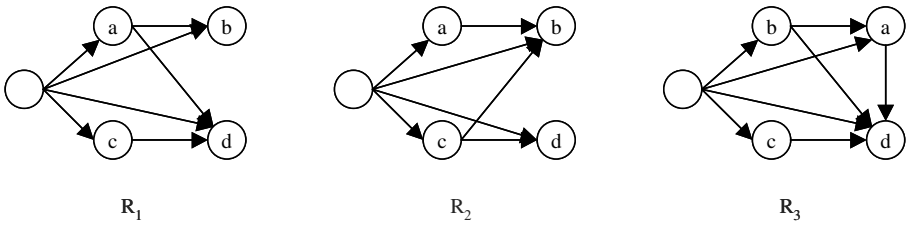

Fig. 3. Graphs of example precedence relations

At this point the relations in $R(L(p))$ may contain pseudo dependencies. Pseudo dependencies are dependencies which actually do not exist in the implicit model. They occur randomly and are caused by random execution times or delays. For example, activities which are embedded in parallel sequences are often performed in a pseudo sequential order due to the fact that they are embedded in opposite ends of their sequences. In order to detect pseudo dependencies in relations we group together all relations which have the same set of activity

occurrences. The main indicator to decide whether a dependency is real or spurious is that a pseudo dependency is a dependency that is not found in every relation of such a group. We use the following algorithm for pseudo dependency detection and elimination.

Algorithm 2. *Let* $\beta : R \to \{o \mid (o, o') \in R \vee (o', o) \in R\}$ *denote a function which returns the set of activity occurrences of a relation* R*. Let* B_j *denote a part of a partition of* $(R(L(p))$ *that is defined by the equivalence relation* $\beta(R) = \beta(R') \Leftrightarrow R \equiv R'$*. Let* $\phi : R \to R'$ *denote a function that computes the transitive reduction of a precedence relation.* U, V *are sets of relations;* u, v *are sets. Input:* $R(L(p))$*.*

For each B_j*{*
  $U := B_j$
  While $|U| > 0$ *{*
    $v := R_1, V := R_1, R_1 \in U$
    $U := U \setminus R_1$
    For each $R_i, R_i \in U$ *{*
      $u := R_1 \cap R_i$
      If $\beta(u) = \beta(R_1)$ *{*
        $V := V \cup R_i$
        $U := U \setminus R_i$
        $v := v \cap R_i$
        }
    }
  $R(L(p)) := R(L(p)) \setminus V$
  $R(L(p)) := R(L(p)) \cup \phi(v)$
  }
}
Return $R(L(p))$

Algorithm 2 returns the adjusted precedence relations $R'(L(p))$ describing each alternative path of executing the process p as it was captured in $L(p)$.

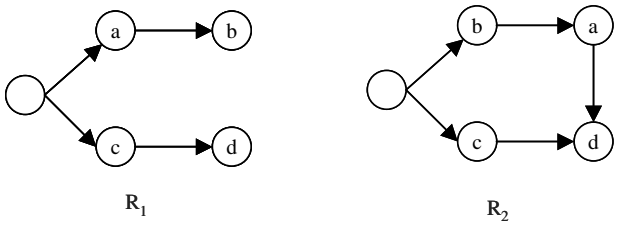

Fig. 4. Graphs of adjusted precedence relations

Example 2. Applying Algorithm 2 on $R(L(p)) = \{R_1, R_2, R_3\}$ from Example 1 results in $R'(L(p)) = \{R_1, R_2\}$ which is depicted in Figure 4. Note that R_1 and R_2 from $R(L(p))$ are merged into R_1 of $R'(L(p))$.

5.4 Model Synthesis

In this step we generate an initial model for p from $R'(L(p))$. First, we generate a sub-model for each path represented by a precedence relation $R_h \in R(L(p))$. Then, a model for the overall workflow is assembled from all sub-models. We use algorithm 3 that handles both task.

Algorithm 3. *Let* $f(R_h) = \{\alpha o_1, o_1 o_2, \ldots, o_{k-1} o_k\}, o_k \in \{o \mid (o, o') \notin R_h \wedge (o', o) \in R_h\}$*, denote a path in* R_h *starting at* α *and ending at an activity occurrence* o_k *that do not have any successor. Let* $F(R) = \{f(R_h)\}$ *denote the set of paths for* $R(L(p))$*. Let* $\eta : f(R_h) \rightarrow \{o \mid ((o, o') \in f(R_h) \vee (o', o) \in f(R_h)) \wedge o \neq \alpha\}$ *denote a function that returns all activity occurrences of a path in ascending order. Let* $\psi : O \rightarrow A$ *be a function that returns an unique activity occurrence reference for an activity occurrence.* sb_n *and* b_m *are variables of type* block*. Input:* $R(L(p))$*.*

For each $R_h, R_h \in R(L(p))\{$
  For each $f_i(R_h), f_i(R_h) \in F(R)$ $\{$
    $sb_i := \mathcal{S}(\psi(o_1), \psi(o_2), \ldots), o_j \in \eta(f_i(R_h)), 1 \leq j \leq |\eta(f_i(R_h))|$
  $\}$
  $b_h := \mathcal{P}(sb_1, \ldots, sb_n)$
$\}$
Return $\mathcal{A}(b_1, \ldots, b_m)$

We apply algorithm 3 on $R(L(p))$. It results an initial model $M(L(p))$ in the form $\mathcal{A}(\mathcal{P}(\mathcal{S}(\ldots), \ldots), \ldots)$.

Example 3. Applying Algorithm 3 on $R'(L(p))$ from Example 2 results in the model $\mathcal{A}(\mathcal{P}(\mathcal{S}(a, b), (c, d)), \mathcal{S}(\mathcal{P}(c, \mathcal{S}(b, a)), d))$ depicted in Figure 2.

5.5 Model Transformation

In this step we transform the initial model into a consolidated and anticipative form. For this purpose we use three different kinds of transformations. First, we detect loops and complete the model with appropriate loop operators. Second, we merge parallel paths in the model by applying a term rewriting system. Third, we change the decision structure of the model in order to make it an anticipatory model.

In section 5.1 we used a labeling operator to artificially differentiate multiple instances of an activity occurrence within a case. Actually, we want to have loop operators containing activity occurrences for those occurrences. At this point we construct loop operators for the labeled occurrences.

Algorithm 4. *Let $\Pi(\mathcal{S})$ denote a minimal partition of the activity occurrence references in a sequence $\mathcal{S}(a, a', \dots)$, so that for each part π the following holds:* $\forall a, a', \psi^{-1}(a) = o \circ i, \psi^{-1}(a') = o \circ j : a \in \pi \Rightarrow a' \in \pi \wedge \forall a, a', a'', a''' \in \pi, \psi^{-1}(a) = o \circ i, \psi^{-1}(a') = o \circ j, \psi^{-1}(a'') = o \circ (i+1), \psi^{-1}(a''') = o \circ (j+1) : a < a' < a'' < a'''$. *Let $\theta : \pi \to (o, o', \dots)$ denote a function that returns the sequence of all activity occurrences in the order in which their first occurrence is found in π, and let $\Psi : (o, o', \dots) \to (\psi(o), \psi(o'), \dots)$ be a function that returns a sequence of activity occurrence references for a sequence of activity occurrences. Let $\xi : A \to \{0, 1\}$ denote a function that returns 1 iff a is a reference for a labeled activity occurrence, and 0 otherwise. Input: $M(p_j)$.*

$\forall S_i \in M$ {
  $a_\diamond :=$ *first labeled a in* S_i
  $a_\triangle :=$ *last labeled a in* S_i
  Split S *into* S_p, S_b, S_p *where*
    $S_p := \mathcal{S}(a, \dots, a'), a' < a_\diamond$
    $S_b := \mathcal{S}(a'', \dots), \forall a : a_\diamond < a < a_\triangle \wedge \xi(a) = 0$
    $S_l := \mathcal{S}(a_\diamond, a''', \dots, a_\triangle), \forall a : \xi(a) = 1$
    $S_s := \mathcal{S}(a'''', \dots), a'''' > a_\triangle$
  Determine $\Pi(S_l)$
    For each $\pi_k, \pi_k \in \Pi(S_l)$ {
      $b_k = \mathcal{L}(\mathcal{S}_k(\Psi(\theta(\pi_k))))$
    }
  $\mathcal{S} := \mathcal{S}(S_p, \mathcal{P}(S_b, b_1, \dots, b_l), S_s)$
}

After constructing loops we merge sequences and loops, respectively, embedded in parallel operators by applying a term rewriting system (TRS_1) that consists of the following rewritings:

$$\mathcal{P}(\mathcal{S}(b_1, \dots, b_u, b_{u+1}, \dots, b_v), \dots, \mathcal{S}(b_1, \dots, b_u, b'_{u+1}, \dots, b_w)) \to$$
$$\mathcal{S}(b_1, \dots, b_u, \mathcal{P}(\mathcal{S}(b_{u+1}, \dots, b_v), \dots, \mathcal{S}(b'_{u+1}, \dots, b'_w))$$
$$\mathcal{P}(\mathcal{S}(b'_1, \dots, b'_u, b_{u+1}, \dots, b_v), \dots, \mathcal{S}(b'_1, \dots, b'_u, b_{u+1}, \dots, b_w)) \to$$
$$\mathcal{S}(\mathcal{P}(\mathcal{S}(b_1, \dots, b_u), \dots, \mathcal{S}(b'_1, \dots, b'_u), b_{u+1}, \dots, b_w)$$

Note, that b denotes a block, i.e. an operator or a term. Because TRS_1 is not confluent we have to specify the order of application. We always apply the first rewriting before the second one if both rewritings are applicable.

Example 4. Let $b_1, b'_1, \dots, b_n, b'_n$ denote references for activity occurrences $b_1, \dots, b_n$. Given the model: $\mathcal{P}(\mathcal{S}(b_1, b_2, b_3, b_4), \mathcal{S}(b'_1, b'_2, b_5, b_6), \mathcal{S}(b_7, b'_5, b'_6))$ Applying TRS_1 produces: $\mathcal{P}(\mathcal{S}(b_1, b_2, \mathcal{P}(\mathcal{S}(b_3, b_4), \mathcal{S}(b_5, b_6))), \mathcal{S}(b_7, b'_5, b'_6))$

Up to now we have mined models from a retrospective perspective. Actually, we want workflow models in order to prescribe the order of executing activities. Therefore, we make a shift from the retrospective perspective to an anticipatory perspective at this point. We do so by splitting the overall *Alternative* operator of $M(L(p))$ into partial *Alternative* operators, and moving any of these operators

to its very latest possible position in its model. For this purpose only, we use the left distributivity of the *Alternative* operator over the *Sequence* operator.

We use the following term rewriting system TRS_2 based on the left and right distributivity of the *Alternative* operator over the *Sequence* operator and the *Parallel* operator.

$$\mathcal{A}(\mathcal{S}(b_1,\ldots,b_u,b_{u+1},\ldots,b_v),\ldots,\mathcal{S}(b_1,\ldots,b_u,b'_{u+1},\ldots,b_w)) \rightarrow$$
$$\mathcal{S}(b_1,\ldots,b_u,\mathcal{A}(\mathcal{S}(b_{u+1},\ldots,b_v),\ldots,\mathcal{S}(b'_{u+1},\ldots,b'_w))$$
$$\mathcal{A}(\mathcal{S}(b'_1,\ldots,b'_u,b_{u+1},\ldots,b_v),\ldots,\mathcal{S}(b'_1,\ldots,b'_u,b_{u+1},\ldots,b_w)) \rightarrow$$
$$\mathcal{S}(\mathcal{A}(\mathcal{S}(b_1,\ldots,b_u),\ldots,\mathcal{S}(b'_1,\ldots,b'_u),b_{u+1},\ldots,b_w)$$
$$\mathcal{A}(\mathcal{P}(b_1,\ldots,b_u),\ldots,\mathcal{P}(b'_1,\ldots,b'_u),\mathcal{P}(b_v,\ldots,b_w),\ldots,\mathcal{P}(b_x,\ldots,b_y)) \rightarrow$$
$$\mathcal{P}(b_1,\ldots,b_u,\mathcal{A}(\mathcal{P}(b_v,\ldots,b_w),\ldots,\mathcal{P}(b_x,\ldots,b_y)))$$

Because TRS_2 is not confluent we have to specify the order of application of its rewritings. Again, we apply the left distribution before right distribution. We use this order because the left distribution has the desired affect of changing points in time decisions have to be made.

Example 5. Let $b_1, b'_1, \ldots, b_n, b'_n$ denote references for activity occurrences $b_1, \ldots, b_n$. Given the model: $\mathcal{A}(\mathcal{S}(b_1,b_2,b_3,b_4,b_5),\mathcal{S}(b'_1,b'_2,b_6,b_6),\mathcal{P}(b_8,b_9,b_{10}),\mathcal{P}(b'_9, b'_{10},b_{11}))$ Applying TRS_2 produces: $\mathcal{A}(\mathcal{S}(b_1,b_2,\mathcal{A}(\mathcal{S}(b_3,b_4,b_5),\mathcal{S}(b_6,b_7))),\mathcal{P}(b_9, b_{10},\mathcal{A}(b_8,b_{11})))$

At this point we have mined a workflow model from a workflow log.

5.6 Complete and Minimal Models

We expect our mining procedure to extract models wich are complete and minimal. Let us summarize how this is achieved regarding the entire process.

Before we start we eliminate inconsistent traces, i.e. the handling of noise is considered to be completed before we run the mining procedure. Based on this, each trace goes into a trace class. In doing so, we ensure that every behavior presented in the log is taken into account and the resulting model is complete. From each trace class exactly one precedence relation is separately extracted. Then, only such precedence relations are merged which differ in eliminated pseudo dependencies. After that, a separate sub-model is extracted from each precedence relation. The term rewriting operating on this model only permits the merging of blocks which are embedded in equal contexts.

We stress on separate handling because it is key in order to mine minimal models. For example, let us consider two simple precedence relations $R_1 = \{(\alpha,a),(a,c),(c,e)\}$ and $R_2 = \{(\alpha,b),(b,c),(c,e)\}$ as extract from a log. In case that we process $R_1 \cup R_2$ instead of processing both relations separately, this results in a model that introduces the possibility of executing e after a and c were executed. But this execution sequence is never found in the log. With our approach the model $\mathcal{A}(\mathcal{S}(a,c,e),\mathcal{S}(b,c,d))$ is constructed. Note that there is no term rewriting or any other transformation that adulterates the sequences.

In case that there are non-desired paths in a model these can be eliminated in the subsequent evaluation stage of the KDD process. For example, such paths may base on very few traces and therefore considered to be irrelevant exceptions. Also, such paths can represent valuable process knowledge. However, the decision about that is not subject to a mining procedure. It is subject to a subsequent model evaluation performed by a user.

6 Engineering the Output

The process mining output comes in form of workflow models based on the meta-model described above. At this point we want to transform these models into models applicable in different workflow systems. A common format for exchanging models between different workflow systems is the WfMC *Interface 1: Process Definition Interchange*[2]. This interface includes a common meta-model for describing process definitions and a textual grammar for the interchange of process definitions, called Workflow Process Definition Language (WPDL). Also, there is a XML version of this language, called XPDL. In order to deploy our models to different workflow systems we can transform them into WPDL-models.

There is a structural difference between our models and WPDL-models in the sense that is WPDL does not support activity occurrence references. Therefore, it is necessary to first insert an additional synchronization of multiple references of the same activity occurrence which are embedded in multiple parallel operators. For this purpose, we expand each reference of an activity occurrence for which there are more than one references embedded in a parallel operator, such that each of these parallel operators contains all such references together. After then, we apply the term rewriting system TRS_1 to the expanded model.

Example 6. Let $b_1, b'_1, \ldots, b_n, b'_n$ denote references for activity occurrences $b_1, \ldots, b_n$. Given the model: $\mathcal{P}(\mathcal{S}(b_1, b_2, \mathcal{P}(b_3, b_4)), \mathcal{S}(b_5, b'_4), \mathcal{S}(b_6, \mathcal{P}(b'_4, b_7)))$ Its expanded form is: $\mathcal{P}(\mathcal{S}(b_1, b_2, \mathcal{P}(b_3, b_4, b_7)), \mathcal{S}(b_5, \mathcal{P}(b'_3, b'_4, b_7), \mathcal{S}(b_6, \mathcal{P}(b'_3, b'_4, b'_7)))$ Applying TRS_1 results in: $\mathcal{S}(\mathcal{P}(\mathcal{S}(b_1, b_2), b_5, b_6), \mathcal{P}(b_3, b_4, b_7))$ Note, that there is now an additional synchronization of b_4 with b_3 and b_7.

After this we can transform a model into WPDL. Algorithm 5 performs a simple transformation.

Algorithm 5. *Let b denote a block of a workflow model. Let* $\phi : B \to T$*, where* $T = \{\mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{L}, \mathcal{O}\}$*, denote a function that returns the type of a Block b. Let* createActivity(T) *be a subroutine that creates and writes a* WPDL Route Activity*, let* createLoopActivity(T) *be a subroutine that creates and writes a* WPDL Loop Activity*, and let* writeActivity(B) *denote a subroutine that writes a* WPDL Activity *for an activity instance b, where* $\phi(b) = \mathcal{O}$*. Let* P, P', D *denote sets of* WPDL Activities*. Let* writeTransitions(P, P') *denote a subroutine that writes* WPDL Transitions *from all* $p \in P$ *to all* $p' \in P'$*, and let* callSelf(B, D) *be a recursive execution of the algorithm. Input:* $b := M$*,* $P := \emptyset$*.*

If $\phi(b) = \mathcal{S}$ {
  For each b' *embedded in* b *ordered by* $\mathcal{S}$ {
    $P := callSelf(b', P)$
    }
  Return P
}
If $\phi(b) = \mathcal{P} \vee \phi(b) = \mathcal{A}$ {
  $D := createActivity(\phi(b))$
  $writeTransitions(P, D)$
  For each b' *embedded in* b {
    $P' := P' \cup callSelf(b', D)$
  }
  $D := createActivity(\phi(b))$
  $writeTransitions(P', D)$
  Return D
}
If $\phi(b) = \mathcal{L}$ {
  $D := createLoopActivity(\mathcal{L})$
  $writeTransitions(P, D)$
  $b' :=$ *the block embedded inside the loop*
  $P := callSelf(b', D)$
  $writeTransitions(P, D)$
  Return D
}
If $\phi(b) = \mathcal{O}$ {
  $writeActivity(\varphi(b))$
  $writeTransitions(P, \varphi(b))$
  Return $\varphi(b)$
}

Besides a transformation of models into WPDL/XPDL, one can provide further algorithms for transforming models into proprietary workflow description languages for particular workflow systems, for example, IBMs *Flow Description Language* (FDL) used by *IBM MQSeries Workflow*.

7 Experimental Evaluation

Our approach is implemented by a tool named *Process Miner*. It is able to read traces of a particular process stored in files of a common XML format or databases and to extract a workflow model based on these traces. The mined workflow models are represented in a graphical editor. With this editor an user can edit the model and export it to a workflow management system. Also, models can be simulated by the tool in order to analyze their performance before deploying them in a workflow management application.

Using a meta-model that consist of *Sequence*, *Parallel*, *Loop* and *Alternative* operators as well as activity occurrence references we have tested the approach on data from different sources. At the one side we used synthetic data. At the other side we used event based data produced by IBM MQSeries Workflow. While executing workflow instances this workflow system logs the events concerning the start and the completion of an activity instance within a particular process instance. The mined models covered the ones which underlies the process instances producing the input data.

8 Related Work

Our approach is close related to the work in [3,4,5,6,7,8].

Mining workflow models was first considered by Agrawal et al. [3,4]. Their approach defines a workflow model as a graph supplemented by conditions for transitions between graph nodes. They divide the mining of workflow models into two problems. The first one is called *graph mining*. The approach presents a solution for this problem in form of an algorithm for extracting a graph from event-based data. The algorithm is based on the key concepts of defining transitive relations between activities and building a graph from it that is transformed into its transitive reduction. In contrast to our approach, they consider an activity to be atomic and mix different different paths of execution. The second problem is about supplementing the graph with conditions in order to distinguish alternative and parallel splits within a workflow model. This problem is called *condition mining*. It is not treated by the approach.

Herbst and Karagiannis deal in their approach with mining workflow models with non-unique tasks names [5,6]. The approach consists of two steps. In the first step, a stochastic task graph is induced from a workflow log. This is done by using a search procedure which embeds a graph generation algorithm in order to find a mapping from activity instances to activity nodes in the graph. The second step transforms the graph into an *ADONIS* workflow model. This step is more extensive than the transformation in our approach because stochastic task graphs do not explicitly outline a synchronization structure, so that this structure has to be extracted in the second step.

In [7,8] van der Aalst and Weijters present an approach on mining workflow nets which is based on counting frequencies of dependencies between activities. The nodes of a workflow net represent activities found in the workflow log. Dependencies between the activities are represented by arcs between the appropriate nodes. In order to decide whether a dependency is represented by the workflow net they use heuristic rules in combination with threshold values. In addition, their approach deals with noisy logs.

There are many similarities between the approaches above and our approach. We consider our approach to be different by the following facts. First, it aims to extract the most specific model for a given workflow log. Second, it explicitly considers time consuming activities instead of atomic activities. Third, it is based on a block-structured meta-model that can be supplemented with application specific basic terms and operators.

9 Summary

In this paper we have presented an approach on mining most specific workflow models from event-based data. We considered process mining a special data mining that requires the development of an appropriate process meta-model and the development of algorithms extracting models based on this meta-model from event based data. According to this we have outlined a block-structured process meta-model consisting of a set of basic operators and supplemented by an basic algebra. Based on this meta-model we described our mining process in detail. In order to deploy mined models we have outlined a simple transformation of workflow models into a common exchange format. Also, we have outlined an overview over the experimental evaluation of our approach and related work.

References

1. Fayyad, U., Piatesky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery in databases. AI Magazine (1996) 37–54
2. WorkflowManagementCoalition: Interface 1: Process definition interchange process model. http://www.wfmc.org/standards/docs/TC-1016-P-v11-IF1-Process-definition-Interchange.pdf (1999) Document Number WfMC TC-1016-P.
3. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Proceedings of the 6. International Conference on Extending Database Technology (EDBT). (1998) 469–483
4. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. IBM Research Report RJ 10100, IBM Almaden Research Center / IBM German Software Development Lab (1998) www.almaden.ibm.com/cs/quest.
5. Herbst, J., Karagiannis, D.: Integrating machine learning and workflow management to support acquisition and adaption of workflow models. In: Proceedings of the Ninth International Workshop on Database and Expert Systems Applications, IEEE (1998) 745–752 IEEE.
6. Herbst, J.: Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen. Dissertation, Universität Ulm (2001)
7. van der Aalst, W.: Process design by discovery: Harvesting workflow knowledge from ad-hoc executions. (In Jarke, M., O'Leary, D., Studer, R., eds.: Knowledge Management: An Interdisciplinary Approach, Dagstuhl Seminar Report Nr. 281) http://aifbhermes.aifb.uni-karlsruhe.de/dagstuhl-km-2000.
8. van der Aalst, W., Weijters, A.: Workflow mining - discovering workflow models from event-based data. In: Proceedings ECAI Workshop "Knowledge Discovery from Temporal and Spatial Data", ECAI 2002, Lyon (22.06.2002) 78–84 http://tasda.elibel.tm.fr/ecai02/w12.pdf.

Evaluation of Correctness Criteria for Dynamic Workflow Changes*

Stefanie Rinderle, Manfred Reichert, and Peter Dadam

University of Ulm, Faculty of Computer Science,
Dept. Databases and Information Systems
{rinderle, reichert, dadam}@informatik.uni-ulm.de

Abstract. The capability to dynamically adapt in-progress workflows (WF) is an essential requirement for any workflow management system (WfMS). This fact has been recognized by the WF community for a long time and different approaches in the area of adaptive workflows have been developed so far. They either enable WF type changes and their propagation to in-progress WF instances or (ad-hoc) changes of single WF instances. Thus, at first glance, many of the major problems related to dynamic WF changes seem to be solved. However, this picture changes when digging deeper into the approaches and considering implementation and usability issues as well. This paper presents important criteria for the correct adaptation of running workflows and analyzes how actual approaches satisfy them. At this, we demonstrate the strengths of the different approaches and provide additional solutions to overcome current limitations. These solutions comprise comprehensive correctness criteria as well as migration rules for change realization.

1 Introduction

A rapidly changing environment and a turbulent market force any company to change their business processes ever more frequently [1]. Process changes become necessary, for example, when new laws come into effect, optimized or restructured business processes are to be implemented, exceptional situations occur, or rapid reactions to a changed market are required. Therefore, a critical challenge for the competitiveness of any enterprise is its ability to quickly react to business process changes [2,3,4].

As pointed out in [2], basically, changes can take place at two levels – the WF type and the WF instance level. Very often changes at the WF instance level are applied in an ad-hoc manner, leading to WF instances with biased execution schema when compared to their original WF schema – in the following, we denote these WF instances as *biased*. Ad-hoc changes become necessary in conjunction with real-world exceptions, e.g., a sudden circulatory collapse of a

* This work was done within the research project "Change management in adaptive workflow systems", which has been founded by the German Research Community (DFG).

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 41–57, 2003.

patient, and they usually affect only *single WF instances*. As opposed to this, in conjunction with schema changes at the WF type level, a collection of related instances may have to be adapted. The challenging question is how to *propagate* WF type changes to running WF instances, but without violating correctness and consistency properties set out by the used WF meta model. In other words, how can we smoothly *migrate* WF instances to a changed WF schema? Additionally, in case of concurrent changes (e.g., concurrent changes at the type and instance level), the exciting question arises how to synchronize them (e.g., how to propagate WF type changes to biased WF instances).

There is a multitude of approaches dealing with flexibility in WfMS [1,2,4,5, 6,7,8]. All of them present very interesting, but partially strongly differing ideas and solutions. Therefore, it is an important job to summarize central criteria for adaptive workflows and to compare actual approaches by using these criteria. Furthermore, we sketch suitable solutions for "still dangling" issues, e.g., related to the problem of checking compliance of WF instances with a modified schema.

At first, we summarize important criteria for different change scenarios, which are necessary to achieve a correct and consistent "post-change"-behavior.

1. **Completeness:** The WF designer must not be restricted, neither by the used WF meta model nor the offered change operations. Therefore, a WF meta model ought to provide a complete set of control and data flow constructs, e.g., allow the designer to model sequences, parallel/alternative branchings, and loops [3]. For practical purposes, at minimum, change operations for inserting and deleting activities as well as control/data dependencies between them are required. Furthermore, it must be able to combine change primitives to define complex changes, e.g., to modify the order of activities.
2. **Correctness:** The ultimate ambition of all adaptive WF meta models must be correctness of dynamic changes [1,2,4,5,6,7,8]; i.e., introducing changes to the runtime system without causing inconsistencies or errors (like deadlocks or improperly invoked activity programs). Therefore, adequate *correctness criteria* are needed. These criteria must not be too restrictive, i.e., no WF instance should be needlessly excluded from applying a dynamic change. Furthermore, it must be clear how the imposed correctness criteria can be easily and quickly checked by the WfMS. This is especially important for large-scale environments with hundreds up to thousands of WF instances.
3. **Change Realization:** Assuming that a dynamic change can be correctly propagated to a WF instance I (along the stated correctness criteria), it should be possible to automatically migrate I to the new schema. In this context, the WF instance state as well as dependent data structures (e.g., user worklists) must be correctly and efficiently adapted.

In the following, we provide a classification of actual approaches which is based on the semantics of the underlying WF meta models and on the above criteria. We point out where the strengths and weaknesses of these approaches lie. To overcome current limitations we discuss solutions which can be easily transferred to other WF models (e.g., a comprehensive correctness criterion, efficient compliance checks, and formal propositions regarding concurrent changes).

In Section 2 we give an overview of current approaches dealing with flexibility in WfMS and classify them with respect to their semantics. Section 3 summarizes and classifies correctness criteria for adaptive workflows. In Section 4 we show how the different approaches realize dynamic WF changes. Section 5 presents a critical discussion and Section 6 closes with a short summary.

2 Approaches Dealing with Flexibility

Figure 1 summarizes adaptive WF meta models which enable a flexible process support. According to [9], we classify those WF meta models according to the evaluation strategies applied for executing WF instances during runtime. The first strategy uses only one type of (control flow) token passing through each WF instance (*True-Tokens*). The other strategy is based on two types of tokens – *True-* and *False-Tokens*. True-Tokens represent activities that are to be executed next and False-Tokens describe activities which have been skipped. Approaches which solely use True-Tokens (cf. Fig. 2) have a *True-Semantics* and include, for example, Petri-Net-based formalisms [2,5,10,1]. Approaches which, in addition, use False-Tokens to represent skipped activities or skipped execution branches can be found in the area of graph-based WF meta models [8,7]. They can be further divided according to the way they represent the True- and False-Tokens. One possibility is to gain these tokens (and therefore the state of running instances) from *execution histories* [4], which log events like start and completion of activity executions (cf. Fig. 3). Alternatively, special (*model-inherent*) markings of activities and/or control edges, which represent a consolidated view on the history logs, can be used [6,7,8] (cf. Fig. 4).

2.1 Approaches with True-Semantics

In [2], a WF schema is represented by a *WF Net* which is a labeled place/transition net $N = (P, T, F, l)$ (cf. Fig. 2). Thereby, P denotes the set of places, T the set of transitions, $F \subseteq (T \times P) \cup (P \times T)$ the set of directed arcs, and l the labeling function, which assigns a label to each transition. The dynamic behavior of a WF instance is described by a *marked WF net* (N, s) with marking (function) s and associated marking rules. The authors abstract from data flow issues, WF attributes and WF resources and consider only one WF instance at a time.

The approach presented in [5,14,10] is based on *Flow Nets*, which are closely related to WF nets. In Chautauqua [14], Flow Nets are generalized to *Information Control Networks (ICN)*. They allow the enactment of a new WF instance by creating an instance specific token, which represents a data form of the enacted ICN. In doing so, data flow is carried out by passing the token through the ICN. WF instances (with same WF type) are distinguished by the use of coloured tokens and are controlled by the same ICN. A meta language to support dynamic evolution of processes is presented in [10]. In the following, however,

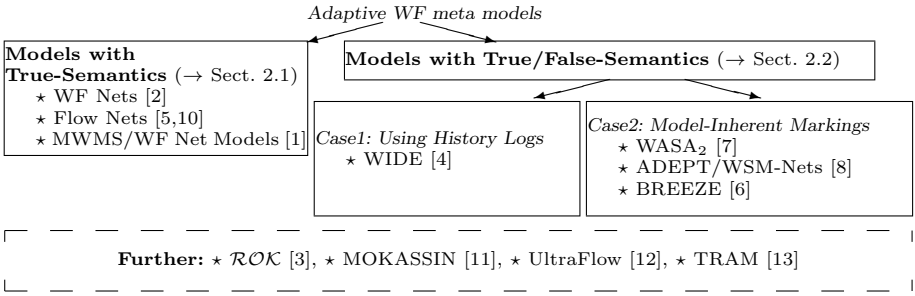

Fig. 1. Selected Approaches Dealing With Flexibility Issues

we focus on the formalisms of Flow Nets as described above. An example for the above approaches is depicted in Fig. 2a.

Another interesting approach is presented by *MWMS* [1]. The authors use *Net Models (NM)*, which are marked, acyclic Free-Choice Petri Nets. Data flow issues are not taken into account. A NM Σ can be mapped to a *Sequential Model (SM)* $\mathcal{A}$ which represents the global states and state transitions of Σ. Thus, $\mathcal{A}$ is comparable to the *reachability graph* of Petri Net Σ.

2.2 Approaches with True/False-Semantics

Case 1: Approaches based on History Logs
The most famous example and also one of the first approaches dealing with dynamic WF changes was offered by *WIDE* [4], which uses a graph-based WF meta model. The modeling of sequential, parallel, alternative, and iterative activity executions is possible. Furthermore, there is a set of global process variables associated with each WF schema S. A WF instance I on WF schema S can be described by S and by its execution history $\mathcal{H} = (\langle \epsilon^S_{I,0}, \mu^S_{I,0} \rangle, \ldots, \langle \epsilon^S_{I,i}, \mu^S_{I,i} \rangle)$, where $\epsilon^S_{I,k}$ denotes the kth completion of a task execution in I and $\mu^S_{I,i}$ denotes related write operations on WF variables performed by $\epsilon^S_{I,k}$.

In WIDE, WF schemata can be described either graphically or by using predecessor and successor functions. Fig. 3 shows the latter variant.

Case 2: Approaches Using Model Inherent Markings
WASA$_2$ [7] introduces an object-oriented WF meta model. It comprises one generic class *Workflow* of which *WF schema* and *WF instance* are instances. Workflows are modeled by using a graph-based WF language comparable to activity nets applied in IBM MQ Series Workflow. In more detail, a *WF schema* $S = (V_S, C_S, D_S)$ is a tuple with sets of activity nodes V_S, control connectors C_S, and data connectors D_S. Similarly, a *WF instance I* can be described. The flow of data is modeled by data connectors which map output and input parameters of subsequent activities. A WF schema S is correct iff all input parameters are correctly supplied by a type-conform output parameter and the graph structure

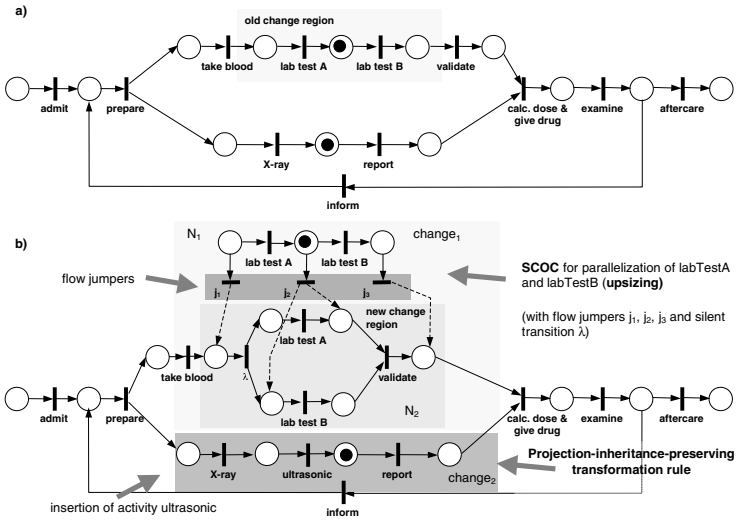

Fig. 2. A Petri-Net-Based Workflow With Changes ([2,5,10])

is acyclic (i.e., no deadlocks will occur). The state of a WF instance is denoted by the marking of the instance nodes (model-inherent).

Another approach with model-inherent markings is based on *Well-Structured Marking-Nets (WSM-Nets)* as applied in our *ADEPT* WfMS [8], for example. WSM-Nets are serial-parallel graphs with distinguishable node and edge types, where loops and branchings are modeled in a block-oriented fashion (block structure). This structure is relaxed by offering *sync edges*, which allow to define precedence relations between activities of parallel branches. We first provide two self-explanatory definitions for WSM-Nets (cf. Def. 1) and for WF instances (cf. Def. 2) based on them.

Definition 1 (Well-Structured Marking-Net, WSM-Nets). *A tuple $S = (N, D, NT, CtrlE, SyncE, LoopE, DataE, EC)$ is called a Well-Structured Marking-Net if the following holds:*

- *N is a set of activities and D a set of process data elements*
- *NT: $N \rightarrow$ {`StartFlow, EndFlow, Activity, AndSplit, AndJoin, XOrSplit, XOrJoin, StartLoop, EndLoop`}*
- *$CtrlE \subset N \times N$ is a precedence relation*
- *$SyncE \subset N \times N$ is a precedence relation between activities of parallel executed branches*
- *$LoopE \subset N \times N$ is a set of loop backward edges*
- *$DataE \subseteq N \times D \times \{read, write\}$ is a set of read/write data links between activities and data elements*
- *EC: $CtrlE \cup SyncE \cup LoopE \rightarrow$ `Predicates`(D) where `Predicates`(D) denotes the set of all valid transition conditions on data elements from D.*

Chemotherapy Treatment WF schema S with
S = ⟨ **Tasks**^{S}, **NullTasks**^{S}, **Vars**^{S}, σ^S, χ^S θ^S ⟩ where
Tasks^S = {'start'$^{(0)}$, admit$^{(1)}$, prepare$^{(2)}$, take blood$^{(3)}$, labTestA$^{(4)}$, labTestB$^{(5)}$, validate$^{(6)}$, XRay$^{(7)}$, report$^{(8)}$, calc. dose & give drug$^{(9)}$, examine patient$^{(10)}$, aftercare$^{(11)}$}
NullTasks^S = ∅, **Vars**^{S} = {bloodVal, xRay, patientData}
Successor function σ^S : 'start' → {admit}, admit → {prepare}, prepare → {take blood, XRay}, take blood → {labTest A}, labTest A → {labTest B}, labTest B → {validate}, XRay → {report}, report → {calc. dose & give drug}, calc. dose & give drug → {examine}, examine → {aftercare, prepare}, aftercare → 'end'
Type function θ^S: {'start', take blood, labTest A, labTest B, XRay, calc. dose & give drug, aftercare} → direct, admit → iterative join, prepare → total fork, {validate, report} → total join, examine → conditional fork
Condition Functiony^S: aftercare → <patient data, 'ok'>, prepare → <patient data, 'not ok'>
WF Instance I on S desribed by execution history
$\mathcal{H}$ = (⟨ start$^S_{I,0}$, <> ⟩, ⟨ admit$^S_{I,1}$, <patientId,256> ⟩, ⟨ prepare$^S_{I,2}$, <> ⟩, ⟨ takeblood$^S_{I,3}$, <> ⟩, ⟨ labTestA$^S_{I,4}$, <bloodVal.resultA,4> ⟩, ⟨ labTestB$^S_{I,5}$, <bloodVal.resultB,15> ⟩, ⟨ xRay$^S_{I,6}$, <XRay,xRay256(1).jpg> ⟩, ⟨ report$^S_{I,7}$, <patientdata.xRay,xRay256(1).doc> ⟩, ⟨ validate$^S_{I,8}$, <patientdata.bloodResults,bloodResults256(1).doc> ⟩, ⟨ calcDoseGiveDrug$^S_{I,9}$, <> ⟩, ⟨ examine$^S_{I,10}$,<patientdata.values,val256(1).doc> ⟩, ⟨ prepare$^S_{I,11}$, <> ⟩, ⟨ takeblood$^S_{I,12}$, <> ⟩, ⟨ labTestA$^S_{I,13}$, <bloodVal.resultA,4.3> ⟩, ⟨ xRay$^S_{I,14}$, <XRay,xRay256(2).jpg> ⟩

Fig. 3. WF Instance Including History Logs in WIDE

A WSM-Net is correct iff
• S_{fwd} = (N, CtrlE, SyncE) is an acyclic graph, i.e., the use of sync edges must not cause undesired cycles leading to deadlocks (for details see [8]),
• for each split (loop start) node there is a unique join (loop end) node, and
• S is structured following a block concept, for which control blocks (sequences, branchings, loops) can be nested but must not overlap.

Definition 2 (WF Instance Based On WSM-Nets). *A WF instance I is defined by a tuple (S, M^S, Val^S, $\mathcal{H}$) where*

- *S = (N, D, NT, CtrlE, SyncE, ...) denotes the WSM-Net the execution of I is based on.*
- *M^S = (NS^S, ES^S) describes node and edge markings of I:*
  NS^S: N → {`NotActivated, Activated, Running, Completed, Skipped`*}*
  ES^S: (CtrlE ∪ SyncE ∪ LoopE) → {`TrueSignaled, FalseSignaled`*}*
- *Val^S is a function on D. It reflects for each data element d ∈ D either its current value or the value* `UNDEFINED` *(if d has not been written yet).*
- *$\mathcal{H}$ = < $e_0, \ldots, e_k$ > is the execution history of I. $e_0, \ldots, e_k$ denote the start and end events of activity executions. For each started activity X the values of data elements read by X and for each completed activity Y the values of data elements written by Y are logged.*

Activities marked as `Activated` are ready to fire and can then be worked on, i.e., their status changes to `Running`. Activities with marking `Skipped` cannot longer be selected for execution. An example of a WF instance based on a WSM-Net is shown in Fig. 4. Another approach using WF graphs similar to WSM-Nets but without loops is offered by *BREEZE* [6].

3 Correctness Criteria for Dynamic WF Changes

We first focus on WF schema changes at the type level and their propagation to running WF instances. Regarding correctness, however, it is not important

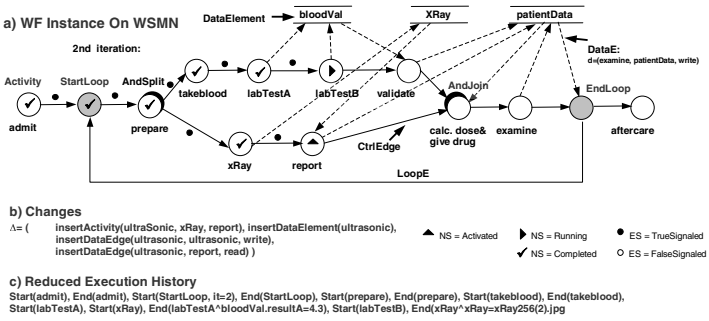

Fig. 4. A (Clinical) WF Instance Based On WSM-Nets

whether a WF type (and therefore a collection of running instances) or a single WF instance is affected by a change. At the end of this section we provide a correctness criterion to handle concurrent changes at the type and the instance level (i.e., to correctly propagate WF type changes to biased WF instances).

In the following, we assume that a WF schema S is always correctly transformed into another schema S' by applying change Δ. What this exactly means depends on the structural and dynamic correctness properties set out by the used WF meta model. We focus on the discussion how the approaches from Section 2 decide whether an instance I can be correctly migrated to a changed schema S' or not. It is remarkable that all discussed solutions are based on formal correctness criteria. While some of these approaches precisely state how to ensure correctness in conjunction with dynamic WF changes, others do not address this point in detail.

We further distinguish between approaches founding their correctness criteria on *graph equivalence* – *WF Nets* [2], *MWMS* [1], and *WASA*$_2$ [7] – and approaches with correctness criteria based on *execution equivalence* – *Flow Nets* [5,10], *WIDE Nets* [4],

and *ADEPT WSM-Nets* [8]. The core idea of graph equivalence is to map the WF instance graph of I to the changed WF schema S'. Depending on the "degree of coverage" it can be decided whether Δ is applicable to I as well. Execution equivalence focuses on the work done by I so far. If this work could have been achieved on S' as well, I can be smoothly migrated to S'. Note that both approaches are closely related to each other.

3.1 Approaches Based on Graph Equivalence

WF Nets: A first representative based on graph equivalence (see above) is described in [2]. The authors use *branching bisimilarity* as an equivalence relation on marked, labeled P/T-Nets (cf. Section 2). It specifies under which conditions two different marked, labeled P/T-Nets S and S' have the same (observable) behavior (notation: $S \sim_b S' \iff \exists$ branching bisimulation $\mathcal{R}$ such that $S\mathcal{R}S'$).

Informally, a marked, labeled P/T-Net must be able to simulate each action of an equivalent marked net.

Correctness-Criterion 1 (Branching Bisimilarity) *Let S be a marked, labeled WF Net and Δ be a change which transforms S into another marked, labeled WF Net S'. Then: Δ can be carried out correctly iff $S \sim_b S'$.*

Generally, it is difficult to ensure Criterion 1 for arbitrary changes. Therefore the authors restrict the set of possible change operations to those which preserve special inheritance relations between the old and the new net (e.g., insertion of activity `ultrasonic` in Fig. 2b). If these inheritance relations hold after applying a change, branching bisimilarity between both nets can be ensured. Inheritance-ensuring change operations are:

- adding sequences as well as parallel, alternative and iterative branches (direction of inheritance from class to subclass) and
- removing sequences as well as parallel, alternative and iterative branches (direction of inheritance from class to superclass).

Unfortunately, branching bisimilarity cannot be automatically ensured for other change operations [2]. The reason for this is a phenomenon called *dynamic change bug*. Examples for non-supported operations are order-changing operations like parallelizing transitions `labTestA` and `labTestB` in Fig. 2a. Excluding those change operations, however, leads to serious problems since related forward and backward jumps have to be frequently applied in practice.

In *MWMS* [1] a set of change operations (parallelization, sequentialization and swapping of activities) is proposed obeying special constraints (summarized by the *Minimal Critical Specification (MCS)* for the underlying SM). Intuitively, only such change operations can be carried out which maintain the given set of activities and which only change their order relations. For these changes the following correctness criterion is provided:

Correctness-Criterion 2 (Safe States) *Let $\mathcal{A} = (S, E, T, s_{in})$ be a SM and Δ be a change operation (within the respective MCS). Δ transforms $\mathcal{A}$ into another SM $\mathcal{A}' = (S', E', T', s'_{in})$. Then an instance I on $\mathcal{A}$ can be migrated to $\mathcal{A}'$ iff I is not in an unsafe state. A state of $\mathcal{A}$ is unsafe iff there is no corresponding state in S'*

Another approach using graph equivalence is offered by *WASA*$_2$ [7]. The author does not explicitly state which changes can be applied. Exemplarily, operations like adding and deleting activities or changing activity orders are provided. The decision whether a change Δ can be applied to an in-progress WF instance or not is based on a *valid mapping* of the *purged WF instance graph* to the WF schema graph. Thereby, the purged WF instance graph is gained by deleting all activities which have not been started yet, and by removing all associated control and data connectors (cf. Fig. 5b). A mapping $m : V_I \nrightarrow V_S$ between a WF instance $I = (V_I, C_I, D_I)$ and a WF schema $S' = (V_{S'}, C_{S'}, D_{S'})$ is defined as follows:

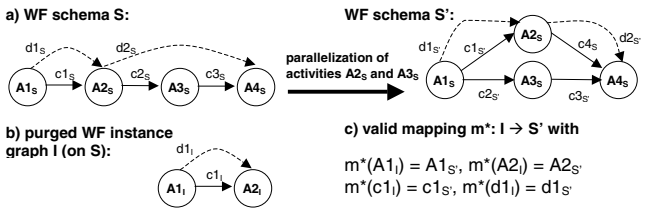

Fig. 5. Valid mapping in WASA$_2$

$$\forall j' \in V_I : \exists j \in V_S \text{ with } [\text{m(j')} = \text{j} \Rightarrow \text{SchemaOf(j')} = \text{j (i.e., j' is based on j)}] \\ \wedge\ [\text{m(j')} = \text{m(k')} \Rightarrow j' = k' \forall j', k' \in V_I]$$

With this, the following correctness criterion based on *valid mappings* between WF instance graph and WF schema graph can be stated:

Correctness-Criterion 3 (Valid Mapping) *Let $I = (V_I, C_I, D_I)$ be a purged WF instance graph derived from WF schema S. Then: Change Δ can be correctly applied to I as well iff $\exists$ a valid mapping m^*: $V_I \mapsto V'_S$. A mapping m^* is valid if all control connectors between two instance objects i, $j \in V_I$ have counterparts i', $j' \in V_{S'}$ with SchemaOf(i) = i' and SchemaOf(j) = j' and $\forall d \in D_I \exists$ d' $\in D_{S'}$ (and vice versa).*

Intuitively, an instance I can be migrated to a changed WF schema S' if each completed activity of I is also contained in S' and all control and data dependencies existing in I have counterparts in S' (cf. Fig. 5c).

To our knowledge no statements have been published so far, how Criterion 3 can be (efficiently) checked. However, an implementation of WASA$_2$ exists [7].

3.2 Approaches Based on Execution Equivalence

Flow Nets: A first approach based on execution equivalence has been presented in [5,10] (details of the used Flow Nets have been given in Section 2.1). In [5], changes of a Flow Net S (with True-Semantics) are carried out by substituting the marked sub-net $\mathcal{N}_1$ of S, which is affected by Δ, by another marked sub-net $\mathcal{N}_2$, which reflects the modifications set out by Δ. Thereby, $\mathcal{N}_1$ is referred to as the *old change region* and $\mathcal{N}_2$ as *new change region*. As the authors point out, the selection of the change regions cannot be fixed. Roughly, the old change region is defined as the smallest marked sub-net containing all activities affected by Δ. Assume, for example, that in Fig. 2a) change operation Δ is to parallelize the so far sequentially ordered activities `LabTestA` and `LabTestB`. Then $\mathcal{N}_1$ is the sub-net containing the affected activities `LabTestA` and `LabTestB`. To be able to decide whether Δ can be correctly propagated to an instance or not the authors introduce the *pre-change (firing) sequence* ω to conceptualize the work done by the WF instance so far. Thereby, ω denotes all transition firings previous to the introduction of Δ. Based on this, the following correctness criterion can be formulated:

Correctness-Criterion 4 (Pre-Change sequence ω) *Let Δ be a change, which transforms Flow Net S into Flow Net S'. Let further be I an instance on S with pre-change sequence ω. Then I can be migrated to S' iff ω can be continued on S' as well.*

In order to check whether Criterion 4 is met, the authors present two kinds of change operations and a special change class, the so called *Synthetic Cut-Over Change (SCOC)*. Applying SCOC, the old change region $\mathcal{N}_1$ is maintained in S' together with $\mathcal{N}_2$ (for an example see Fig. 2b); i.e., S' contains two versions of the modified subnet. How this "fusion" of old and new change region is carried out depends on the applied change. In [5] two change scenarios – *Upsizing* and *Downsizing* – are introduced. Upsizing means that $\mathcal{N}_2$ can "do more" than $\mathcal{N}_1$, i.e., the set of all valid firing sequences on $\mathcal{N}_1$ is a subset of all valid firing sequences on $\mathcal{N}_2$. Downsizing is the dual counterpart of upsizing, i.e., $\mathcal{N}_2$ can "do less" than $\mathcal{N}_1$. For example, Fig. 2b shows an upsizing. In this case, the SCOC can be constructed by sticking $\mathcal{N}_1$ and $\mathcal{N}_2$ together over *flow-jumpers* (cf. Fig. 2b). Flow-jumpers are transitions, which map each marking of $\mathcal{N}_1$ to a marking of $\mathcal{N}_2$. This way of constructing the SCOC in conjunction with upsizing operations is correct regarding Criterion 4. In the other case – downsizing – the SCOC is constructed by merging $\mathcal{N}_1$ and $\mathcal{N}_2$ over one output place, i.e., instances with tokens in $\mathcal{N}_1$ are further executed according to the old net. Trivially, this restrictive approach is also correct regarding Criterion 4. Other important change operations, like the insertion of new activities, are not discussed. Very interesting is that upsizing and downsizing are excluded by [2] since these changes lead to the dynamic change bug (cf. Section 3.1).

A widely-used correctness property is the *compliance criterion* introduced by *WIDE* [4]. Intuitively, change Δ of WF schema S can be correctly propagated to a WF instance I iff the execution of I, taken place so far, can be "simulated" on the modified WF schema S' as well. Note that Criterion 5 is similar to Criterion 4 at first glance. But Criterion 4 is only based on a snapshot of the WF execution whereas Criterion 5 takes the whole WF execution into account. Since the authors work with a history-based execution model, compliance is based on *replaying the execution history* $\mathcal{H}$ of WF instance I on the changed WF schema S'. Formally:

Correctness-Criterion 5 (Intuitive Compliance Criterion) *Let S be a WF schema and I be a WF instance on S with execution history $\mathcal{H}$. Let further S be transformed into another schema S' by change operation Δ. Then: I is compliant with S' iff $\mathcal{H}$ can be produced on S' as well.*

Assume that in WF schema S in Fig. 3 task `aftercare` is to be deleted. Referring to execution history $\mathcal{H}$ of WF instance I (cf. Fig. 3) the intended deletion is possible since $\mathcal{H}$ contains no entry related to `aftercare` and can therefore be (re-)produced on the changed WF schema as well. As opposed to this, the insertion of a new task `ultrasonic` between tasks `xRay` and `report` is not possible regarding Criterion 5. The reason is that activity `ultrasonic` has not written

any entries into $\mathcal{H}$ during the first loop iteration (note that the loop is actually in its 2^{nd} iteration). Inserting `ultrasonic` in the actual loop iteration, however, would cause no inconsistencies or errors at runtime (irrespective of (rare) roll-back operations into former loop iterations which become more expensive). Consequently, Criterion 5 is too restrictive, especially in conjunction with iterative, long-running workflows. Since in [4] no further information about how to check Criterion 5 is given, we assume that compliance is ensured by trying to replay the whole execution history on the changed WF schema. Doing so causes a big overhead due to the possibly extensive volume of the history (caused by information like user assignments or time stamps).

WSM-Nets: In *ADEPT* [8], we focus on finding a correctness criterion which works in conjunction with loops as well as other orthogonal aspects (e.g., data flow). The key to solution with respect to loops is to be able to differentiate between completed and future executions of loop iterations. From a formal point of view there are two possible approaches. One approach is to logically treat loop structures as being equivalent to respective linear sequences. The other approach is to maintain the loop construct but to restrict the evaluation to the relevant parts of the execution history (cf. Def. 3). We adope the second approach since it facilitates the treatment of nested loops, provides a good basis for implementation, and leads to "smart" proofs.

Definition 3 (Reduced Execution History $\mathcal{H}_{red}$). *Let I be a WF instance with execution history $\mathcal{H}$. The reduced execution history $\mathcal{H}_{red}$ is obtained as follows: In the absence of loops $\mathcal{H}_{red}$ is identical to $\mathcal{H}$. Otherwise, it is derived from $\mathcal{H}$ by discarding all history entries related to other loop iterations than the last one (completed loop) or the actual iteration (running loop). (Note that $\mathcal{H}_{red}$ can be easily produced in conjunction with nested loops as well.)*

As an example take Fig. 4c, which shows the reduced execution history for the instance from Fig. 4a. Taking Def. 3 we now present a comprehensive compliance criterion for WF schema evolution. According to this property, an instance is compliant with a changed schema iff the reduced execution history can be produced on the modified schema as well.

Correctness-Criterion 6 (Comprehensive Compliance Criterion) *Let I be a WF instance on WF schema S with execution history $\mathcal{H}$ and reduced execution history $\mathcal{H}_{red}$. Assume further that a change Δ transforms S into the correct WF schema S'. Then I is said to be compliant with S' iff $\mathcal{H}_{red}$ can be produced on S' as well.*

Again, the challenging question is how to efficiently check the comprehensive compliance criterion. We present easily and quickly checkable marking conditions for each kind of change on WSM-Nets [8] (additive, subtractive, order-changing, and complex operations). Due to lack of space, we exemplarily summarize these conditions for additive change operations in Theorem 1.

Theorem 1 (Additive Change Operations On WSM-Nets). *Let S = (N, D, ...) be a correct WSM-Net and I be a WF instance on S with execution history*

$\mathcal{H}_{red}$. Assume further that change Δ transforms S into correct WSM-Net S' = (N', D', ...).
(a) Δ inserts an activity n_{insert} (with associated control and sync edges) into S. Then:
I is compliant with S' $\Leftrightarrow$
$\forall\ n \in \{x \in N \mid n_{insert} \rightarrow x \in E'\}$: *NS(n)* $\in$ {`NotActivated, Activated, Skipped`} $\vee$
n_{insert} is inserted into an already skipped branch of an XOr-branching
(b) Δ inserts a control edge $\mathtt{n}_{src} \rightarrow \mathtt{n}_{dest}$ into S. Then:
I is compliant with S' $\Leftrightarrow$ NS(n_{dest}) $\in$ {`NotActivated, Activated, Skipped`}
(c) Δ inserts a sync edge $\mathtt{n}_{src} \rightarrow \mathtt{n}_{dest}$ into S (n_{src} and n_{dest} ordered parallel so far). Then:
I is compliant with S' $\Leftrightarrow$
[*NS(n_{dest})* $\in$ {`NotActivated, Activated, Skipped`}] $\vee$
[*NS(n_{src})* = `Completed` $\wedge$ *NS(n_{dest})* $\in$ {`Running, Completed`} *with*
$\exists e_i$ = `END(`n_{src}`)`, e_j = `START(`n_{dest}`)` $\in \mathcal{H}_{red} \wedge i < j$))] $\vee$
[*NS(n_{src})* = `Skipped` $\wedge$ *NS(n_{dest})* $\in$
{`Running, Completed`}) *with*
$\forall\ n \in N_{critical}$ *with NS(n)* $\neq$ `Skipped`:
$\exists e_i$ = `START(`$\mathtt{n}_{dest}$`)`, e_j = `END(n)` $\in \mathcal{H}_{red}$ *with* $j < i$),
where $N_{critical}$ = (c_pred(S, n_{src})* $\neg$ *c_pred*(S, n_{dest}))*
and c_pred(S, n)) denotes all direct/indirect predecessors of n in S concerning*
edges $\in$ *CtrlE*]

For additive change operations, Theorem 1 presents precise conditions for efficient compliance checks with an estimated complexity of $O(n)$. These conditions base on a consolidated view of the reduced execution history $\mathcal{H}_{red}$. As an example take the insertion of activity `ultrasonic` as defined by change Δ in Fig. 4b. According to Theorem 1(a) it is only necessary to determine the marking of the successors of `ultrasonic` in S'. In our example, activity `report` is marked as `Activated` such that Δ can be applied to the WF instance depicted in Fig. 4a. To show their efficiency we have implemented several simulations checking our compliance conditions.

Besides, we explicitly deal with compliance issues in conjunction with data flow changes. We shortly summarize the basic ideas: Data elements can be always inserted, but must not be deleted if there was a read or write access on them. Read data edges $e_{read} = (n, d, read)$ on data element d can only be inserted or deleted iff activity n is marked as `NotActivated, Activated` or `Skipped`. Write data edges $e_{write} = (n, d, write)$ on data element d can only be inserted or deleted iff activity n has not been completed yet.

3.3 Concurrent Type and Instance Changes

Finally, we want to give an idea how the propagation of WF schema changes to biased WF instances can be managed correctly in the context of WSM-Nets [8].

But it should be clear that the following conclusions are applicable to other WF meta models as well. To meet a formal point of view, we first give a definition of a biased WF instance (compare Def. 1 and 2).

Definition 4 (Biased WF Instance). *A biased instance I is described by a tuple (S, Δ_I, $M^{S+\Delta_I}$, $Val^{S+\Delta_I}$, $\mathcal{H}$), where S denotes the WSM-Net from which I was created and Δ_I comprises ad-hoc changes op_I^1, ..., op_I^n that have been applied to I so far. WSM-Net $S_I := S + \Delta_I$, which results from the application of Δ_I to S, is called execution schema of I.*

Comparable to the already discussed correctness criteria we introduce a general criterion that allows us to argue about both – propagation of WF schema changes on "normal" (unbiased) and on biased WF instances. Obviously, when propagating a WF schema change Δ_S to a biased WF instance I we must not only consider its current state (i.e., marking $M^{S+\Delta_I}$) but we also have to cope with structural and semantic conflicts that may exist between the concurrent changes Δ_I and Δ_S. (Note that both, Δ_I and Δ_S have been based on S.) Due to lack of space we only consider structural conflicts in the following.

Correctness-Criterion 7 (Concurrent Changes) *Let S be a correct WSM-Net and I = (S, Δ_I, $M^{S+\Delta_I}$, ...) be a biased WF instance that was created from S. Let further Δ_S be a change operation, which transforms S into another correct WSM-Net S'. Then: Δ_S may be propagated to biased WF instance I :⇔*

1. *$S^* = (S + \Delta_I) + \Delta_S$ is a correct WSM-Net, i.e., Δ_S can be correctly applied to the execution schema $S_I = (S + \Delta_I)$.*
2. *I is compliant with S^*; i.e., the reduced execution history $\mathcal{H}_{red}$ can be produced on S^* as well. The marking M^{S^*} resulting from this is considered as a correct marking.*

Again, the challenging question is how to efficiently verify the conditions set out by Criterion 7. A naive solution would be to first generate the WSM-Net $S_I + \Delta_S$ and then to check whether it satisfies the required structural and dynamic properties. Generally, this would be too expensive, in particular if different WF aspects (control flow, data flow, work assignments, etc.) are concerned or Δ_S is to be propagated to a large collection of instances. Instead we must define appropriate and efficient rules for excluding potential conflicts (e.g., undesired cycles and deadlocks) between instance and type changes for as many instances as possible. Due to lack of space we abstain from further details.

4 Change Realization

We have now reached the stage of checking compliance of WF instances with a changed WF schema. This analysis leads to two instance categories – compliant and non-compliant WF instances [4,6]. We first discuss how the different approaches concretely carry out the migration of compliant WF instances

(*instance adaptations*). Then a short overview about approaches dealing with non-compliant WF instances is presented.

Approaches With True-Semantics: For Petri-Net based approaches, instance migration means to find a suitable marking on the changed net.

WF Nets: In [2], for each imposed change operation *transfer rules* are defined, which automatically adapt net markings. Concerning the insertion of sequences and alternative branches, the respective transfer rule maps marking s of the old net S to the identical marking on the new net S' (transfer rule is identity function $id : (S, s) \mapsto (S', s)$). As an example take the markings of the net in Fig. 2 before and after insertion of transition `ultrasonic`. For other change operations the insertion of additional tokens becomes necessary. Examples are changes like the insertion of new parallel branches or the deletion of alternative branches and sequences which contain tokens. Due to lack of space we abstain from discussing further transfer rules.

Flow Nets: For the change operations provided by [5,10], trivially, markings are adapted by constructing the SCOC (cf. Section 3.2). Note that the schema resulting from a SCOC always contains the marking of the old net.

Approaches With True/False-Semantics: To our knowledge neither *WASA*$_2$ [7] nor *WIDE* [4] provide detailed information about marking adaptations. In WIDE, however, follow-up markings may result from the replay of the execution history. As mentioned in Section 3.2, doing so is very expensive since execution histories often contain extensive data.

WSM-Nets: Our *ADEPT* approach is somewhat different regarding marking adaptations of compliant instances. To keep these adaptations efficient, we restrict them to those nodes and edges of the respective execution schema S_I, which constitute the context of the change region. Therefore, for each change operation *op* initial sets of nodes end edges to be re-evaluated are determined. Depending on the result of the evaluation the inspection of additional nodes and edges may become necessary. In addition, we benefit from well-defined marking rules as well as the way markings are represented (preserving markings of passed regions, True/False semantics). As an example take change Δ in Fig. 4. In the course of the following adaptation, `ultrasonic` has to be marked as `Activated` and marking of `report` is re-evaluated to `NotActivated`. Finally, an algorithm has been formulated, which evaluates instance markings with an estimated complexity of $O(n)$.

Dealing With Non-Compliant WF Instances: There are several approaches dealing with (temporarily) non-compliant instances [5,6]. *BREEZE* [6] provides a special graph construct which consists of compensation activities. With this, non-compliant instances are partially rolled back into a compliant state. The first approach which gives an idea of *delayed migration* is presented in the area of *Flow Nets* [5]. As an example consider Fig. 2. Even if the given instance passes through the old change region, a delayed migration to the new change region is possible when another loop iteration takes place. We have adopted this concept and suggest to keep such (temporary) non compliant instances *pending to migrate*.

5 Discussion

WF Nets: In [2], a wide range of change operations is covered and provided with correctness criteria and (automatic) transfer rules for adapting markings. In case of selected change operations (e.g., adding new parallel branches) new tokens can be automatically created when migrating instances. Though the authors completely abstract from data flow, for practical purposes it is necessary that tokens carry data flow information as well. Therefore, the semantics of newly inserted tokens at runtime is not always clear.

Flow Nets: [5,10] introduce a special class of changes, the described SCOC (cf. Section 3.2). Trivially, applying SCOC changes, marking adaptations are always correctly performed since the old change region is completely contained in the new net. For a special kind of changes – upsizing – the states of the old change region are mapped to states of the new change region by flow-jumpers (cf. Fig. 2b). [10] suggest to determine these flow jumpers manually which implies a very experienced WF designer. A very nice idea is offered by delayed migrations, i.e., the possibility of (temporarily) non-compliant instances to later migrate to the changes schema, e.g., when a loop back takes place. This approach gets even more complex when data flow issues are to be taken into account as well.

In *MWMS* [1] a special class of change operations is offered which provides correct migration of instances in safe states. Both the imposed WF model and the offered change operations are strongly restricted. To our knowledge, there is no detailed discussion about how to check the provided correctness criterion and how to adapt markings after instance migrations.

WASA$_2$ [7] suggests valid mappings between the purged WF instance graph and the changed WF schema in order to preserve correctness. Though [7] presents an implementation there is no detailed conceptualization of the mentioned valid mappings. In WASA$_2$ the loop problem is not present since only acyclic WF graphs are allowed. However, WASA$_2$ is one of the few approaches, which benefits from a concrete implementation of a powerful WF engine.

WIDE [4] has offered a cornerstone for many other approaches – the intuitive compliance criterion. Unfortunately, this criterion suffers from its restrictions concerning loops (cf. Section 3). Furthermore, it is not clear how the given criterion can be checked and implemented. If we had assumed that replaying the whole execution history is necessary this could not be efficiently realized. Generally, execution history logs are not captured in primary storage and contain extensive information like work assignments, time stamps, etc.

All discussed approaches – except our *ADEPT* approach – do not explicitly deal with data flow aspects. Furthermore, one of the few approaches to care about orthogonal aspects in conjunction with dynamic WF changes is offered by *BREEZE* [6]. Here, WF schema evolution in conjunction with time management is discussed. As already mentioned concurrent changes have been adressed only in the area of our *ADEPT* WSM-Nets so far.

Table 1 compares the discussed approaches along the stated criteria.

Table 1. A Comparison Of The Discussed Approaches

	WF Nets	*Flow Nets*	*MWMS*	*WASA$_2$*	*WIDE*	*ADEPT*
Completeness of						
• WF Model	–	○	–	–	+	+
• Changes	–	+	–	+	+	+
• Correctness Criteria	+	+	+	+	–	+
Checking Compliance	+	–	–	–	–	+
Change Realization	+	–	?	?	+	+
Available Implementation	?	+	?	+	?	+

6 Summary and Outlook

In this paper we have compared actual approaches dealing with adaptive workflows along fundamental criteria. Thereby the main focus lies on providing correctness criteria to decide whether a WF instance can be smoothly migrated to a changed WF schema or not. In many applications, the question how to efficiently check these criteria, how to accomplish instance migrations, how to implement the presented concepts, and how to offer change facilities to users remains unanswered. Therefore, we have presented simple state conditions for compliance checks and a nice solution to adapt instance markings after change propagation. Furthermore, we have discussed issues regarding concurrent changes. We strictly encourage other research groups to deal with this exciting problem as well and to provide implementations of their concepts within a powerful WF engine. There are many other interesting questions mainly concerning implementation of the presented concepts. Within this, questions related to change authorization, change analyses, and usability have to be carefully answered.

References

1. Agostini, A., De Michelis, G.: Improving flexibility of workflow management systems. In: Proc. BPM '2000. LNCS 1806, Springer (2000) 218–234
2. van der Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoretical Computer Science **270** (2002) 125–203
3. Edmond, D., ter Hofstede, A.: A reflective infrastructure for workflow adaptability. Data and Knowledge Engineering **34** (2000) 271–304
4. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data and Knowledge Engineering **24** (1998) 211–238
5. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: Proc. Int'l Conf. on Org. Comp. Sys. (COOCS '95), Milpitas, CA (1995) 10–21
6. Sadiq, S., Marjanovic, O., Orlowska, M.: Managing change and time in dynamic workflow processes. Int'l J Coop IS **9** (2000)
7. Weske, M.: Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In: Proc. 34th Hawaii Int'l Conf. on System Sciences (HICSS-34). (2001)

8. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. Journal of Intelligent Inf. Systems **10** (1998) 93–129
9. Kiepuszewski, B., ter Hofstede, A., Bussler, C.: On structured workflow modelling. In: Proc. CAiSE '00. LNCS 1789, Springer (2000) 431–445
10. Ellis, C., Keddara, K.: A workflow change is a workflow. In: Proc. BPM 2000. Volume 1806 of LNCS., Springer (2000) 516–534
11. Joeris, G., Herzog, O.: Managing evolving workflow specifications. In: Proc. Int'l Conf. on Coop. Inf. Systems (CoopIS '98), New York City (1998) 310–321
12. Fent, A., Reiter, H., Freitag, B.: Design for change: Evolving workflow specifications in ULTRAflow. In: Proc. CAiSE '02. (2002) 516–534
13. Kradolfer, M., Geppert, A.: Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In: Proc. CoopIS '99, Edinburgh (1999) 104–114
14. Ellis, C., Maltzahn, C.: The Chautauqua workflow system. In: Proc. 30th Int'l Conf. on System Science, Maui (1997)

Integrated Business Process Management: Using State-Based Business Rules to Communicate between Disparate Stakeholders

Donald C. McDermid

School of Computer and Information Science
Edith Cowan University
Perth, Australia
d.mcdermid@ecu.edu.au

Abstract. We need to put more emphasis on managing the communication between different types of stakeholders and in particular we need to use diagrammatic constructs that support that communication process. This paper describes a state-based approach to capturing business rules that has been tested with different stakeholders in several business process applications. The examples provided in this paper show the benefits of using this notation as a means of communicating between three different groups of stakeholders.

1 Introduction

"*We ran into trouble because we didn't know how to manage what we had, not because we lacked the techniques*." [23] These prophetic words of Aron in 1969 in those early days of discovery about the software development process still ring true today. Not only do they still apply to software engineering generally, but it will be argued in this paper that they also apply to integrated business process management, especially if we have to communicate with different groups of stakeholders in an organisation. Figure 1 summarises the overall scope of business process management as far as this paper is concerned. It is asserted that all stakeholders in business process management have to be considered because otherwise there will be important gaps and omissions in the discussion and this is highly likely to lead to the failure of the business process implementation.

In order for business processes to be successfully implemented, it is argued that three distinct groups of stakeholders need to be involved. These are the business people or users themselves, the systems analysts and the programmers (figure 1). Note that (perhaps uncharacteristically) analysts and programmers have been

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 58–71, 2003.
Springer-Verlag Berlin Heidelberg 2003

separated into different groupings. These groupings have been chosen because each group has a distinct ontology and thus worldview and vocabulary about what is important to them [3]. They have different goals and objectives and the organisation measures them by different criteria.

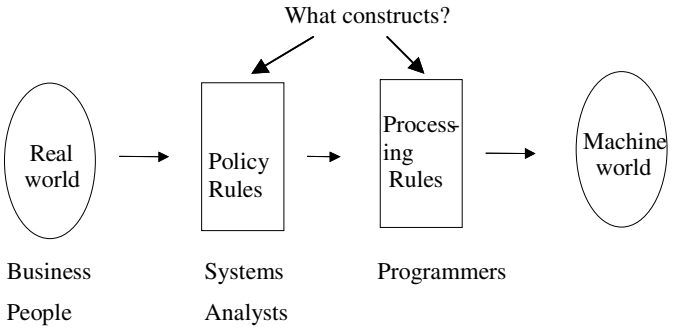

Fig. 1. Mapping of Scripts from the Real World to the Machine World (after [29])

Business people work in a relatively unstructured world compared to analysts. Their worldview and vocabulary are not necessarily bound by the need to work with specific well-typed constructs such as events, conditions or whatever. They have become accustomed to leaving the task of formalizing that process to the 'experts' i.e. the systems analysts. And so the question arises as to what construct might best facilitate that process of formalizing informal statements. The term *structured* is used here in preference to the term semi-formal and also to avoid confusion with the term formal which is often reserved for mathematical specifications [7]. As far as systems analysts are concerned, informal representations are unacceptable as a starting point to proper business process definition since they are ambiguous, open to assumptions or contain omissions.

The next interface to be considered is that between systems analyst and programmer. In addition to the job title of programmer, they may be named software engineer, designer, analyst/programmer and so on. Their prime concern is or ensuring an effective and efficient working artifact. They take no responsibility for the correctness or completeness of the business process specification. That is the job of the systems analyst. Yet, importantly, they take many decisions that affect the final running of a computer-based information system. They take decisions often on navigational issues, error back-out rules; often they have to elaborate the rather bare business rules specified by the analyst. Yet again, the question arises as to what constructs best serve the process of communication between systems analysts on the one hand who are (relatively speaking) more concerned about whether the right system is being built (validation) and on the other the programmers who are more concerned with whether a working system has been built that meets that specification (in that context verification).

This paper will argue that a notation based on states and which is based on Petri nets is a good way of serving the needs of integrated business process management. This argument is dervied from detailed studies that focused on the information and communication needs between disparate stakeholders, though unfortunately space limitations do not permit detailed discussion of this aspect. While much of what follows is necessarily technical in nature, discussion will continue to relate back to the overall management and communication issues introduced above.

In the next section a definition of a business rule and also an introduction to the rationale for the four main constructs notation of a diagram that captures business rules is provided. This is followed by a section that describes the technique itself, in particular the steps for producing a model of the business process and its rules. Sections 2, and 3 deal with the communication interface between users and systems analysts. In section 4, the interface between systems analyst and programmer is discussed and examples provided of the kind of communication and specification issues that arise. The paper ends in section 5 with a brief summary.

2 Business Rule Definition and Constructs

The definition of a business rule used here evolved over the course of the author's doctoral research [9]. It contains a list of the constructs of a business rule namely states, events, conditions and signals. *States* reflect the status of an object of interest at any given time, so for example a manufacturing work order might occupy the states unstarted, in progress or completed. *Events* are actions carried out internally by the organisation and therefore under the control of and owned by the organisation. They are considered to be instantaneous occurrences which reflect the organisation's policy on what should happen in a particular circumstance eg cancel work order. One important role of the event is to *avoid* describing processing detail. Rather, such detail is best kept separate from 'policy' rules as discussed later. *Conditions* define the criteria by which objects of interest in the business move from one state to the next as business events take place. Sometimes many conditions require to be met in order for an event to take place thus increasing complexity. It is argued that modelling conditions without the context of states and events (and vice versa) is far less powerful. Lastly, *signals* either enter or leave the human activity system. Signals which enter the system will typically initiate activity within the system and so these are called *triggers*. Triggers may be external such as a customer sending an order or internal such as one department sending a document to another department which then triggers off some activity. Further, a trigger may be a time trigger eg an activity beginning at the start of the day or the end of the month. Those signals which leave the system serve the purpose of informing those outside the system of what has occurred inside the system and therefore are referred to as *messages*. Thus, though some might argue that the idea of a condition is at the heart of a business rule (as in [10]'s definition), the related constructs of state, event and signal provide a context for the business rule. While none of the constructs are new, the packaging of them into a business rule is unique and argued to be necessary for communication amongst stakeholders.

The definition of a business rule is as follows, though at this stage the reasoning for some of the specific wording may not be clear. *A business rule is an explicit state change context in an organisation which describes the states, conditions and signals associated with events that either change the state of a human activity system so that subsequently it will respond differently to external stimuli or reinforce the constraints which govern a human activity system.*

One significant aspect of a business rule is the idea of a state change context in a business rule. This is a unique solution in business rules models ie one not seen in other definitions. Most other definitions of a business rule focus on conditions and *do not* involve the state as a construct (eg [11]). In this definition states are those anchors upon which the business rule is modelled. Without states it much more difficult to put a boundary on a business rule. In other words, the state change context makes business rules modelling more tractable and therefore easier for communication across disparate stakeholders.

Figure 1 also distinguished business policy rules and business processing rules. Business policy rules reflect the essence or core of what the business is about. For example, a bank would want to distinguish overdraft customers from customers with a positive balance. They will typically be treated differently and different processes and procedures would be likely to apply. Clearly any human activity system associated with bank customers would have policies for different types of customers and what happens to them. Also, observe that such rules would have to contain the pre-conditions which fire processes or activities in the business eg overdraft customers are given higher interest rates, reminder letters etc.

At a lower level of abstraction, what may be termed business processing rules can be seen to exist. These rules arguably are also business rules. Perhaps they are better described as elaborations of business policy rules. They would contain the detail of how the processes and procedures are actually carried out *as well as* the condition(s) under which the business policy rule is executed. So, for example, a rule expressing the exact calculation of a sales tax or the detailed sequence of steps in accepting an order would be classified as a business processing rule. Though business processing rules are elaborations of policy rules, they can be distinguished in the following way. A rule is a business policy rule if it *only* contains the necessary information for an external observer (ie someone outside the human activity system but associated with it) to detect that the system has changed state. In other words, business policy rules only describe the minimum necessary to *determine* state changes (ie the external WHAT) whereas business processing rules describe the detailed execution of state changes (ie the internal HOW). Business processing rules therefore require a notation, possibly a formal notation, which defines precisely the detail of the state change. Constraint diagrams and action contracts [12, 13] would appear to have the richness and formality necessary to specify business processing rules though these rely on objects rather than states as is the case here. Of course, business policy rules are deemed more acceptable as an initial model of business processes by the very fact that they deliberately hide detailed processing aspects, thus allowing dialogue with users to be conducted at an 'appropriate' level – again underlining the assertion that clear communication is the uppermost consideration.

3 Constructing the Business (Policy) Rules Diagram

In this section, for brevity, the term *business rule* will refer only to those high level policy rules in a human activity system. (Later section will discuss business processing rules.) Such a model would be a conceptual model of business policy and as far as possible therefore be unconstrained and independent of processing considerations. The section describes how to model business processes using the constructs defined above and culminates in a set of use cases (populated by the constructs discussed above) that collectively specify a major business process. At this point each use case is called a User Business Rules Diagram (UBRD) and for brevity, this set of use cases (UBRDs) is referred to as the Business Rules Diagram (BRD). The steps involved in creating the BRD are:

- identify candidate business rules (policy rules),
- identify candidate business events and signals,
- identify candidate objects in problem situation,
- construct object life history for each candidate object identified,
- construct User Business Rules Diagrams

3.1 Identify Candidate Business Rules (Policy Rules)

This is achieved by assembling groups of users together and generating a list of candidate rules (eg by brainstorming or any other technique discussed earlier). Afterwards, the list is reviewed and rules which are obviously not policy rules are removed. Table 1 contains a set of policy rules for an order processing case which will be used throughout this paper for illustration purposes. Though every reasonable attempt is made to identify all policy rules, later steps provide an opportunity to identify rules which have been overlooked here.

Each event and signal at this stage is only a candidate. A business event should be a significant occurrence in a business process (ie externally verifiable). The event therefore should have impact on how some component in the system is subsequently dealt with. Again, it is not vital that every event or signal is identified here, as subsequent steps may throw up more events. This step could be combined with the first step into a single session in which two lists are generated. Thus some structure is being added to the informality.

3.2 Identify Candidate Business Events and Signals

A first-cut list of events and signals is also achieved through a brainstorming/review process. At this stage, the distinction between an event and a signal may not be so clear in the minds of users. Both events and signals are considered instantaneous and

further the exact boundary of the information system may still have to be clarified in detail. In the first instance it is considered more important to identify through brainstorming as complete a list as possible and then that list can be classified into events, triggers or messages. Triggers, such as the receipt of an order from a customer are coded with a 'T' for trigger. Events such as the creation of an outstanding order item (or back order) due to insufficient stock are coded with a 'E' and messages such as the sending of an invoice are coded with an 'M'. Of course the act of coding may throw up omissions in the brainstorming process. For example, the sending of an invoice (ie a message) is a different activity from the creation of the invoice (which is an event) and thus the list may be added to as omissions are identified. Table 2 contains a list of candidate business events and signals for the same order processing sample case.

Table 1. Candidate Business Rules for Sample Case

Orders sent by mail or telephone
Omission on order line leads to deletion of that order line
Credit balance >= order value to accept order, otherwise reject
Stock qty >= order qty for normal order, otherwise outstanding
One invoice for one order
Sum of payments = order value - sum of credit notes
One order may have many credit notes
Many payments per invoice possible
Overdue invoices occur 30 days after Statement
If product line not carried, reject item
If unobtainable multiples reject item
New order created for outstanding items
Only good customers may obtain credit orders
Credit balance reduced for all items on an order including outstanding items

3.3 Identify Candidate Objects in Problem Situation

Object modelling is a way of partitioning a system into components [15]. Such partitioning is performed to enable individuals working with the system to deal with properties and aspects of the system at a more local and focussed level (ie at the object level) as opposed to working with the whole system. For example the object customer may be perceived to be a component of an order processing system. The ability to focus on aspects of a customer without reference to a whole business process which involves customers makes it easier for users to explicate requirements. Working with objects is considered by many to be natural and intuitive to users [8].

Table 2. Candidate List of Business Events and Signals.

Receive customer order	T
Delete line	E
Reject order	E
Create new order	E
Send invoice	M
Generate credit note	E
Receive payment	T
Create outstanding item	E
Create new customer	E
Move to good customer	E
Move to bad customer	E

Objects may be simple or complex. A simple object is one in which each property is single-valued; objects whose properties are multi-valued or whose properties are themselves objects are considered complex [16]. In modelling a business in terms of objects, there is a need to model complex objects (as is illustrated shortly) to allow working at the level that users regard as appropriate.

[8] refer to the difference between the real world and a model as a semantic gap and claim that object-oriented models make for small semantic gaps. The advantage of this correspondence between real world and model is that users and analyst(s) then have a common framework in the model to clarify issues concerned with for example completeness and correctness. The analyst thus *learns about the human activity system* through a dialogue with users about the model.

By scanning the lists of events, signals and rules, a candidate list of objects can be identified. From the two tables above the candidate objects of *order, stock* and *customer* can be identified. Less clear is the existence of another object called *outstanding order item***.** Assume that the way this sample case operates is that outstanding order items exist over some time and during that time several may accumulate from across several orders for the same customer. Such a situation would justify identifying a separate object and importantly also lead to rewriting the business rules list and updating it with a new rule ie that an order can be created from outstanding items from several different initial orders. Note also that in turn, this ought to lead to updating the events list so that outstanding items are at some point converted to normal order lines. At any rate, for the purpose of illustration here, there are four objects ie order, stock, customer and outstanding order item.

An object life history (OLH) is modelled by identifying the sequence of states that an object occupies over time. [19] defines the state of a system at a moment in time as '*the set of relevant properties which that system has at that time. Any system has an unlimited number of properties. Only some of these are relevant to any particular research.... The value of the relevant properties constitute the state of the system*'. Thus the state of a system (and therefore the state of an object which is some component of a system) is an abstraction of the system to meet a particular end. For

the BRD, candidate states are those which are occupied for some period of time and which are *perceived by an external observer as relevant for describing the business rules of the system*. Because states can be validated by external stakeholders, this makes communication clearer and provides a solid context for clear unambiguous specification. While the OLH can show the sequence or route that any one instance of an object may take, it also is a template indicating all possible routes of all instances of an object (class).

3.4 Construct Object Life Histories for Each Object Identified

Figure 2 contains the object life histories for the four candidate objects. The practice of modelling object or entity life histories is relatively well established in the computing community [15, 17, 18] though it is by no means ubiquitous. However, this practice usually relates to constructing OLHs from simple objects rather than complex objects. Entities by definition are simple.

States in the OLH are shown as circles and sequence (precedence) is shown by a single-headed arrow. Note the existence of the deleted state for outstanding order item. On first inspection this may seem strange in the sense that non-existence is being modelled rather than some aspect of existence. Also it might be questioned why this aspect is modelled for one object but not the others. The problem is that outstanding items are to be 'converted' to normal order lines at some point. While the logic for this has not yet been identified at this point, identifying a deleted state is one way of flagging that this has to be dealt with later.

Ideas from the work of David Harel and his higraphs are used in the BRD. Variously referred to as Statechart [20], Higraph [21] and Objectchart [22] his diagrams incorporate a powerful device known as Harel depth for reducing the visual complexity of certain diagrams. The basic principle involves using the concept of depth to delimit the scope of effect of an arc. In figure 3, an object order is shown which has two states - a telephone order state and a mail order state. Arc b represents a direct connection to the mail order state and arc d a connection away from the mail order state. Arcs b and d therefore allow the logic relating to a specific state of an object to be depicted. On the other hand arcs a and c respectively show connections to and from the whole object. In other words, these arcs apply equally to all states within the object, in this case the telephone order state and the mail order state. Though simple and intuitive this is a powerful mechanism for reducing the number of arcs necessary on a graph and thus makes the graph more readable.

There are two ways in which Harel depth (sometimes called the Harel blob) is used in the OLH. The first way is essentially that described above ie where it is desired to indicate selection. A softbox is drawn around, in this case, the two states of telephone order and mailed order and the label 'either' inserted. See figure 2. Arrows stop and start at the edge of the softbox indicating that either (but not both) state may exist for any one instance of an order and that any predecessor and successor states apply to both. The second way is where there is a need to indicate parallel states. Take the order OLH for instance. Here, credit-note order (which means that at least one credit-note has been generated for that order instance), part-paid order and overdue order

may all be created provided that the instance is in invoiced order state. However, an important difference here is that these three states may co-exist ie it is possible that credit-notes may have been generated while at the same time part-payments have been received and the order becomes overdue. The 'parallel' label indicates the propensity for parallel existence. See figure 2.

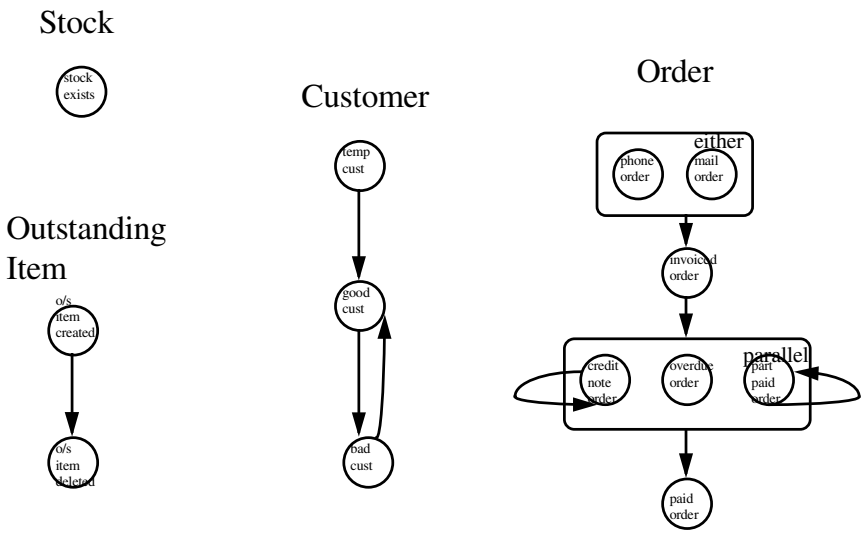

Fig. 2. OLHs for Candidate Objects

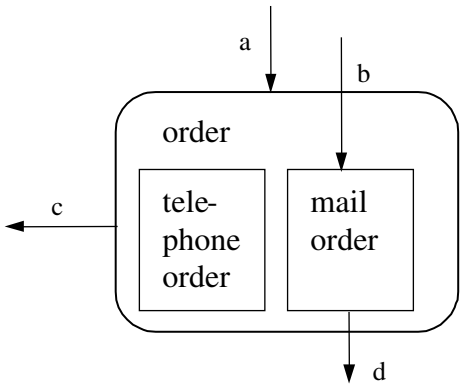

Fig. 3. Illustration of Harel Depth

3.5 Construct User Business Rules Diagram

The analyst scans the OLHs to identify use cases. [8] use the term 'use case' to describe an episode of use of an information system. A use case represents one session of interaction with the system; such a session is a meaningful, whole task in its own right as far as a user of the system is concerned. For example, the episode of receiving an order and processing it to its acceptance is a use case. One user usually will perform one use case (though one use case type may be performed by many users). There are many possible outcomes to a use case. For instance, in processing an order the possible outcomes might include outright acceptance, outright rejection, deletion of some ordered lines but acceptance of the rest and so on. For the most part, one state change (or state creation) will equate to one use case though not always. For each use case identified, the analyst then assembles the user or users who will have responsibility for that use case. A User Business Rules Diagram (UBRD) is drawn for each use case.

The User Business Rules Diagram is constructed by elaborating the states in the OLHs diagram by adding conditions (ie the candidate business rules), events, triggers and messages from tables such as tables 1 and 2. The notational constructs used are defined in figure 4.

The purpose of drawing the UBRD is for the systems analyst to glean as much information as possible about the business rules concerning the use case. It is envisaged that the analyst will create the UBRD with the users present and that therefore it is important that the notational complexity of the UBRD is kept to a minimum. Figure 5 represents the most complex UBRD in the case study.

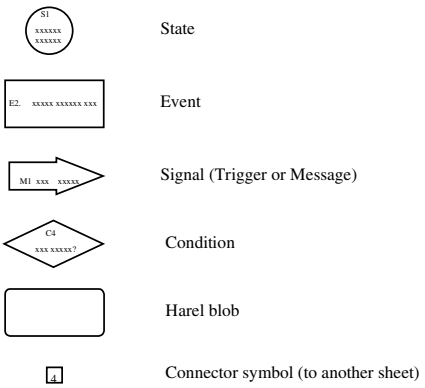

Fig. 4. Notational Constructs for Business Rules Diagram

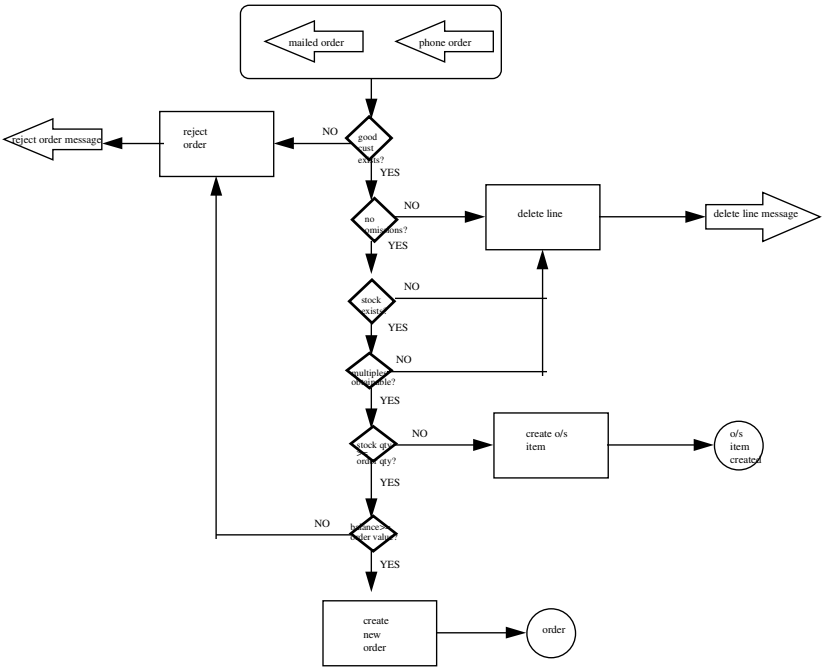

Fig. 5. Accept Order UBRD

Note the 'flowchart look and feel' of the UBRD in figure 5. This keeps the diagram intuitive for the users' benefit yet at the same time allows the analyst to build a comprehensive picture of the rules behind a use case. Notice that some outcomes such as the creation of the outstanding item state do not involve the sending of a message to the customer. Another situation also arises in figure 5 wherein a deleted line message is sent to the customer but no state change occurs. Overall this permits the range of business policy situations to be described in an efficient way.

4 Constructing the Business (Processing) Rules Diagram

In this section the translation of policy rules into processing rules is discussed. Remember that the significant difference between the two is that the purpose of the policy rule is that it describes policy at a sufficiently high level of abstraction that users and analysts can engage in meaningful discussion about the business separate from computer-based considerations (such as navigation of a website). By contrast, the purpose of processing rules is for the systems analyst and programmer to engage in discussion about more computer-based detail. So it is necessary for processing rules to be able to describe the kinds of detail necessary for implementation of

computer-based systems. Two examples of the type of additional specification information required for processing rules are given below.

The first relates to validation and is typical of situations where business rules are defined at a high level of abstraction by the business people and the computer people are often left to sort out the detail. Suppose in a bank ATM (Automatic Teller Machine) system, the customer has to enter a PIN (Personal Identification Number). The policy rule for this may well be specified at quite a high level such as '*customers must enter a valid PIN; if the PIN is not valid, processing does not proceed and the card is rejected with an error message*'. In this situation it has been left to the computer people to work out exactly what the sequence of entry of data will be and deal with what should happen if the PIN is entered incorrectly and in particular how many times a customer may re-attempt to enter the PIN. From a modelling standpoint, what is happening is that the rules are being elaborated or extended by describing additional detail. So it is necessary for the technique to be able to allow itself to be *easily and seamlessly extended* at this stage.

The second example relates to the need to preserve navigational integrity within business processes (and reflects the assumption therefore that many business processes will end up as extranet or intranet applications). One of the biggest challenges in website design has been to give users the ability to move freely and quickly within a website with the minimum number of 'clicks' yet at the same time preserve integrity in terms of only allowing authorised access to information. In poorly designed sites, those inclined often find ways to bypass access rights. So from a processing rules modelling standpoint, it would be important be able to ensure that this does not happen. Clearly, states are an excellent vehicle for this as only users in an acceptable state will get access to a function or to information.

The above examples represent the kind of practical design concern faced in developing a notation for processing rules. To-date, processing rules have been modelled across a number of different types of business processes including an electronic journal website, facilities management and customer relationship management [24, 25, 26, 27 and 28] and later this year this completed work will be the subject of a doctoral thesis of which the author is a supervisor. Two aspects of this research are relevant here. Firstly, the overwhelming evidence from these tests is that concept of using state as the anchor upon which to model processing rules continues to be sound and workable. In addition, the constructs of event, condition and signal are all fundamental to specifying processing rules. The second aspect is that to describe processing rules adequately, there is a need to distinguish 'business' states from 'processing' states. Processing states are effectively 'sub-states' of business states in that they allow us to define states that are important in tracking progress within the computer-based function (e.g. a validation algorithm) but not in themselves of direct interest to the business function. Thus they have no existence outside of the function in which they exist i.e. they have no persistence, whereas business states do.

5 Summary

This paper has presented an argument for the overriding need for different stakeholders to be able to communicate with each other across the whole domain of business process development. Too often have researchers concentrated on only part of the process (e.g. the modelling component or the software architecture component) and this has resulted in a piecemeal, ineffectual approach. Secondly this paper has described a technique using state-based business rules to capture and model the detail of business processes. Importantly, it has been demonstrated that the main constructs of this technique are used seamlessly throughout the whole domain of business process development as a means of communicating between stakeholders. Thus this approach can be seen as a means to integrate business process development.

In this paper, no new design constructs have been introduced. All have been in the literature for some time and will be well understood by researcher and practitioner alike. However, the thrust of this paper has been that both managerial issues (especially in terms of managing the communication between groups) and technical solutions (in terms of choosing the most appropriate constructs to facilitate communication) need to be blended together in a way that makes the whole process is viable. In a sense, communication is that biggest challenge we face today in terms of achieving successful business process management and we need the maturity to understand how the very rich set of technical wisdom we have (as a discipline) accumulated over the last forty years can be applied to solve management issues such as stakeholder communication. As stated in the first sentence of this paper, "*We ran into trouble because we didn't know how to manage what we had, not because we lacked the techniques*." [23]

References

1. Greenspan, S., Mylopoulos, J., and Borgida, A., (1994). 'On Formal Requirements Modeling Languages: RML Revisited. *Proceedings of the 16th International Conference on Software Engineering.* (pp135–147) Los Alamitos, IEEE Press.
2. Vitalari, N. P. (1992). 'Structuring the Requirements Analysis Process for Information systems: A Proposition Viewpoint'. In W. W. Cotterman, and Senn J. A. (eds), *Challenges and Strategies for Research in Systems Development* (pp163–79): John Wiley, Chichester, England.
3. Vidgen, R. (1997). 'Stakeholders, soft systems and technology: separation and mediation in the analysis of information systems requirements.' *Information Systems Journal* 7, pp21–46.
4. Darke, P., and Shanks, G. (1997). 'User viewpoint modelling: understanding and representing user viewpoints during requirements definition.' *Information Systems Journal* 7, pp213–239.
5. Leifer, R., Lee, S. and Durgee, J. (1994). 'Deep structures: real information requirements determination'. *Information and Management, 27*, pp275–85.
6. IEEE (1984*). Std-830*. New York, IEEE.

7. Pohl, K. (1994). 'The Three Dimensions of Requirements Engineering: A Framework and Its Applications.' *Information Systems* 19, pp243–258.
8. Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. (1992). *Object-oriented Software Engineering: A Use Case Driven Approach*. Wokingham, England: Addison-Wesley.
9. McDermid, D. C. (1998). *The Development of the Business Rules Diagram*, PhD Thesis, Curtin University of Technology, Australia.
10. Loosley, C. (1992). 'Separation and Integration in the Zachman Framework'. *Data Base Newsletter, 20*(1).
11. Appleton, D. (1988). 'Second Generation Applications'. *Database Programming and Design*(Feb), pp48–54.
12. Kent, S. and Gil, J. (1998). 'Visualising action contracts in object-oriented modelling'. IEE Software, Vol 145 (2–3), pp70–78.
13. Kent, S. (1997). 'Constraint diagrams: Visualising invariants in object-oriented models'. Proceedings of OOPSLA97. ACM Press.
14. Jones, B. (1991). Letter to editor. *Database Programming and Design*, Nov. p9.
15. Coad, P., and Yourdon, E. (1991). *Object-Oriented Analysis*. (2nd ed.). Englewood Cliffs, New Jersey: Yourdon Press.
16. Atknison, M., Banchilhon, F., DeWitt, D., Dittrich, K., Maier, D. and Zdonik, S. (1989). 'The object-oriented database system manifesto', *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*. Kyoto, Japan.
17. Downs, E., Clare, P., and Coe, I. (1988). *Structured systems analysis and design method.* Hemel Hempstead, England: Prentice Hall.
18. Shlaer, S., and Mellor, S. J. (1992). *Object Lifecycles: Modeling the World in States*. Englewood Cliffs, New Jersey: Yourdon Press.
19. Ackoff, R. (1971). 'Towards a system of system concepts'. *Management Science, 17*, pp661–71.
20. Harel, D. (1987). 'Statecharts: A Visual Formalism for Complex Systems', *Science of Computer Programming* 8, pp231–274.
21. Harel, D. (1988). 'On Visual Formalisms'. *Communications of the ACM, 31*(5), pp514–30.
22. Coleman, D., Hayes, F. and Bear, S. (1992). 'Introducing Objectcharts or How to Use Statecharts in Object-oriented Design'. *IEEE Transactions on Software Engineering, 18*(1), pp9–18.
23. Thomsett, R. (1993). *"Third Wave Project Management: A Handbook for Managing the Complex Information Systems for the 1990s",* Yourdon Press Computing Series, Prentice-Hall, New Jersey.
24. Johnstone, M N, McDermid D C and Venable J R (2002), "Shared Use of Diagrams in Requirements Elicitation: Roles, Expectations and Behaviours", *Proceedings of the 13th ACIS*, Melbourne, Australia, 2002
25. Johnstone, M N, McDermid D C and Venable J R (2002), "Modelling E-Business Security Requirements: Developer and Client Expectations", AWRE 2002, Melbourne, Australia.
26. Johnstone, M N and McDermid D C (2001), "Using Ontological Ideas to Facilitate the Comparison of Requirements Elicitation Methods", *Proceedings of the 12th ACIS*, Coffs Harbour, NSW, Australia, 2001
27. Johnstone, M N, McDermid D C and Venable J R (2000), "Teaching an Old Dog New Tricks: Modelling Electronic Commerce with Business Rules" In: Gable, G. and Vitale, M. (eds), *Proceedings of the 11th Australasian Conference on Information Systems*.
28. Johnstone, M N and McDermid D C (1999), "Extending and Validating the Business Rules Diagram Method", *Proceedings of the 10th ACIS*, Wellington, New Zealand
29. Wand, Y., and Weber, R. (1993). 'On the Ontological Expressiveness of Information Systems Analysis and Design Grammars'. *Journal of Information Systems, 3*, pp217–37.

Structuring Business Objectives: A Business Process Modeling Perspective

Dina Neiger and Leonid Churilov

School of Business Systems, Faculty of Information Technology, PO Box 63B, Monash University, Victoria 3800, Australia
{Dina.Neiger, Leonid.Churilov}@infotech.monash.edu.au

Abstract. Business process modeling assists an enterprise to achieve strategic objectives by providing methodology and tools to develop integrated business process models, however there is little discussion on *how to identify the strategic objectives* or *how process and functional objectives link to the strategic objectives*. On the other hand, decision-modeling literature offers well-established techniques for identification of strategic objectives and associated objectives networks but lacks discussion of *the processes and functions aimed at achieving these objectives*. That neither discipline can provide business with an overall solution is the main problem addressed by this paper through development of a formalized model for linking the two sets of objectives. The resulting model is used to analyze the links between different types of objectives and as a basis for a practical implementation procedure for business modeling. Concepts discussed in the paper are illustrated within a Human Resources context.

Keywords: Business process modeling, Design and analysis of business processes, Decision analysis, Business objectives, Human resources

1 Introduction and Motivation

Integrated enterprise architecture frameworks enable an enterprise to comprehensively describe its processes, associated objects and flows providing the organization with tools to enable effective planning, analysis and design of its functions and structures [16]. Architecture of Integrated Information Systems (ARIS) developed by Scheer [23], [24], [25] is one of the widely adopted and well-established integrated enterprise architecture frameworks [7], [16]. At the core of the ARIS representation of the business is a descriptive representation of business *processes* based on a functional view. Within this framework the functions are grouped into processes which are further grouped into organizational value added chains aimed at achieving overall organizational objectives [14], [23], [24], [25]. The goals associated with each level of functional hierarchy are included in the functional view and can also be represented using an Objective Diagram that allows the modeler to define a network of organizational goals [24], [10]. There is an agreement in business process modeling literature (e.g.

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 72–87, 2003.

[7], [23], [24]) that the definition of organizational objectives is the first step in business process modeling, however usually there is little discussion of what this involves.

The definition of organizational objectives is also an important first step in business *decision* modeling (e.g. [6], [11]) and, unlike *process* modeling, it is accompanied by well-established techniques for identifying and structuring objectives using "value-focusing thinking" methodology developed by Keeney [11]. As this methodology was developed for and is applied primarily within the decision modeling context, *functions* and *processes* required to achieve these objectives unless explicitly encapsulated within a given decision making routine such as decision trees, influence diagrams, Markov chains, mathematical programming etc. (e.g. [6], [32]) are rarely considered.

The problem in continuing separation between two approaches is that real-world businesses are not split along process and decision modeling lines therefore it is essential to establish how *process* and *decision* modeling approaches interact with each other in order to provide an overall solution. While enterprise modeling researchers intuitively link processes and objectives (e.g. [35]), a formal model integrating these approaches through the application of the "value-focused thinking" methodology to define an objectives structure within the process-modeling context, would result in more effective process and decision modeling tools. By drawing upon the extensive research in the modeling methods discussed, this paper addresses the following objectives:

- develop and illustrate a formal linkage model between decision and process modeling approaches to structuring business objectives;
- analyze the relationship between different levels of objectives;
- suggest guidelines for practical application of the "value-focused thinking" methodology to business process modeling; and
- discuss how the application of the "value-focused thinking" methodology impacts upon the interpretation of existing process modeling tools.

The paper is organized as follows: Section 2 briefly discusses the definition of business objectives within the "value-focused thinking" and process-modeling frameworks. In Section 3 the formalized linkage model is introduced and is illustrated in Section 4 within a Human Resource context. This is followed by the implementation guidelines in Section 5. In Section 6 the impact of the proposed methodology is discussed and the paper concludes with a brief summary of the findings.

2 Background

In this section we introduce definitions and modeling approaches adopted by the decision and process modeling disciplines for describing business objectives.

2.1 Objectives Definition within "Value-Focused Thinking" Framework

Within the classical decision analysis-based "value focused thinking" framework an objective is defined as: "a statement of something that one wants to strive toward"

([12] p. 34). The objectives are categorized into two types: fundamental objectives that "concern the ends that decision makers value in a specific decision context" and means objectives that "are methods to achieve ends" ([12] p. 34). To assist with the identification and structuring of objectives Clemen and Reilly ([6] p. 49) summarize the rules for describing the structure of objectives relationships as follows:

Table 1. How to construct means-objectives networks and fundamental-objectives hierarchies as appears in Clemen and Reilly [6] page 49 Figure 3.3.

	Fundamental Objectives	Means Objectives
to move:	*downward in the hierarchy:*	*away from fundamental objectives:*
ask:	What do you mean by that?	How could you achieve this?
to move:	*upward in the hierarchy*:	*toward fundamental objective:*
ask:	Of what more general objective is this an aspect?	Why is that important?

As further illustrated in Section 4, the objectives structure that results from the application of these definitions, rules and associated procedures for eliciting required information incorporates fundamental organizational values [6], [12]. Consequently, each objective within the structure is defined within the context of the overall organizational objectives facilitating a holistic approach to problem solving by decision makers. Development of an objectives hierarchy is the first step in understanding conflicting objectives and developing value trade-offs to feed into a quantitative decision model such as a Multi-Criteria Decision Model (MCDA) [19]. The benefits of applying this method in real world organizations have been well-documented [6], [9], [12], [13], [18], [33] and include a "simple structure for decision makers to understand and use" ([18], p.1), "uncovering hidden objectives, guiding strategic thinking, improving communication" ([33], p. 4), "high benefit-to-effort ratios", "new decision opportunities" ([13]), etc.

The relative drawback of the methodology is that it does not stipulate how the means network is translated into the objectives hierarchy or how different levels of the means network are distinguished. As the business process provides the mechanism for transformation between different levels of business objectives it should be possible to overcome most of these problems by tighter integration of links between fundamental and means objectives with the organizational processes.

2.2 Objectives Definition within ARIS Framework

The "ARIS house of business engineering" developed by Scheer [24] provides tools to describe a consolidated business model through different views of an extended event-driven process chain (e-EPC). The latter is based on a concept of an event-controlled driven process chain (EPC) [20], [24] first developed by "SAP AG …to represent business processes in a simple, yet, clear manner" ([14], p. 149). For the purposes of this paper, we are primarily concerned with the *process view* that describes business

processes by combining functions, events, goals and application software components of the business model.

Within the ARIS Methods Manual [10] goals are implicitly defined through the definition of the function as "a technical task or action performed on an object to support one or more company goals" ([10] p.4-1). *This approach assumes that company goals and objectives are known to the modeler in advance and are supported by functions [25].*

The terms "function" and "process" are used interchangeably and synonymously by Scheer ([25] p. 21) with processes generally described as complex functions at the top level of the functional tree, and which can be divided into sub-functions to reduce complexity ([10] p. 4-2). The lowest level of the functional tree consists of elementary functions defined as "functions which cannot be divided any further for the purpose of business process analysis" ([10] p. 4-2). Process modeling rules for structuring goals corresponding to business processes and functions can be summarized as follows ([25] p. 22): goals can be linked with one another by means of a directed network; functions are able to support multiple goals; and the association between functions and goals can be inherited by higher levels.

The main advantage of the integrated process-modeling approach to goal specification is that it positions each goal into the overall business context by directly linking the goal to the relevant function. The discussion of how these goals are to be derived and relate to the overall goals, however, is limited to the suggestion ([25] p. 22) that "goals can be derived by critical success factors as developed by Rockart." As critical success factors are also derived from organizational goals [22], this does not resolve where organizational goals come from, how should they be structured and what their relationship is to process and functional goals.

The relationship between "value-focused" objectives and process goals is discussed in the next section.

3 Linkage Model

This section includes: a formal description of an EPC; modifications to this description to include objectives and corresponding links; and formalized links between process modeling and "value-focused" objectives.

Formal description of an EPC. Keller and Teufel [14] p. 159 provide a formal description of the EPC as a 7-tuple $EPCM = (Id, \nu, \kappa, \tau, \tau_\kappa, \alpha, \alpha_\kappa)$ where:

- Id is a unique identifier of an EPC model described by $EPCM_{Id}$.
- ν is a non-empty, finite set of nodes of an EPC with $|\nu|$ denoting cardinality of ν. As an EPC consists of at least one starting and one ending events, and one function it follows that $3 \le |\nu| < \infty$.
- κ is a link relationship, which describes the connections between the various types of nodes, κ is defined as $\kappa \subseteq \nu \times \nu$.

- τ, τ_κ are representations that assign a type to every node or link.
- α, α_κ are representations that assign attributes to every node or link type.
- (x,y) is a link that connects nodes x and y iff $x \to y$, where x refers to the start node and y refers to the end node and $\forall x, y \in \nu : (x, y) \in \kappa$

Modifications to the EPC description. The types of nodes included in the EPC description are limited to $\tau : \nu \to \{\text{function, event, connectors, functional objective}\}$, where the function type includes elementary and hierarchically ranked (or complex) functions that are part of the process represented by the EPC in question. Other types of nodes and links are excluded from this description because they either do not relate to objectives or have the same relationship to objectives as one of the included elements (e.g. functions and hierarchically ranked functions). Therefore the formalism in this section can be extended to a full e-EPC model without loss of information.

The notation above is used to describe the following sets: functions and functional objectives (equation 1), processes and process objectives (equation 2), means and fundamental objectives (equation 3), and corresponding links (equations 4, 5, 6, 7).

Functions and functional objectives: Let F_{Id} be a finite non-empty set of functions within an EPC and O_f be a finite non-empty set of functional objectives:

$$F_{Id} := \{f \in \nu \setminus \tau(f) = \text{function} \wedge f \in EPCM_{Id}\} \qquad (1)$$
$$O_f := \{o_f \in \nu \setminus \tau(o_f) = \text{functional objective}\}$$

Process and process objectives. Let P be a set of processes, where each process represented by an EPC is defined as a member of a set of all subsets (power set) of the set of functions within that EPC consistent with the definitions in Section 2.2. Let O_p be a finite non-empty set of process objectives distinct from the set of functional objectives within that process.

$$P := \{p : \forall Id \; p \in power(F_{Id}) \wedge p \neq \emptyset\}; \quad O_p := \{o_p : \text{process objective}\} \qquad (2)$$

Process objectives and the process are included in another EPC or a value-added chain (VAD). According to Scheer [24], the EPCs and VADs are combined into a process hierarchy with the EPCs at the lowest level of the hierarchy including only elementary functions, the VADs at the top level consisting only of the processes, and all other levels being a mixture of elementary functions and processes.

Means and fundamental objectives. Let O_m, O_d be finite, non-empty sets of means and fundamental objectives independent of the process model.

$$O_m := \{m : \text{means objective}\}, O_d := \{d : \text{fundamental objective}\} \qquad (3)$$

Links. Directional links describing control flow within an EPC and simple links describing assignment of objectives to functions and process are included in EPC link types. Each function and process is linked to at least one objective.

$$\tau_\kappa : \kappa \to \{\text{control flow link, objectives assignment link (Oal)}\} \quad (4)$$
$$\forall f \in F_{Id}\ \exists o_f \in O_f : \exists(f, o_f) \subseteq \kappa \wedge \tau_\kappa((f, o_f)) = \text{Oal}$$
$$\forall p \in P\ \exists o_p \in O_p : \exists(p, o_p) \subseteq \kappa \wedge \tau_\kappa((p, o_p)) = \text{Oal}$$

Let μ be a finite non-empty set of objectives flow links describing links between objectives. These links are directional to allow objectives structures to be represented as directional networks. Note that the structure of relationship between means objectives is represented as a directed network whereas the fundamental objectives structure is represented as a hierarchy [6].

$$\tau_\mu : \mu \to \{\text{objectives flow link (Ofl)}\} \quad (5)$$

These links are subject to the following constraints:

- In a fully described process, no process objective can be fulfilled without at least one function contributing to it and each function must contribute to at least one process objective.

$$\forall o_f \in O_f\ \exists o_p \in O_p \wedge \forall o_p \in O_p\ \exists o_f \in O_f : \exists(o_f, o_p) \subseteq \mu \wedge \tau_\mu(o_f, o_p) = \text{Ofl} \quad (6)$$

- Similarly, rules in Table 1 imply that every means objective must link to at least one fundamental objective to describe what needs to be done for a fundamental objective to be fulfilled; and every fundamental objective must link to at least one means objective to describe why a means objective is important.

$$\forall o_m \in O_m\ \exists o_d \in O_d \wedge \forall o_d \in O_d\ \exists o_m \in O_m : \exists(o_m, o_d) \subseteq \mu \wedge \tau_\mu(o_m, o_d) = \text{Ofl} \quad (7)$$

Links between process modeling and "value-focused thinking" objectives. The links between these types of objectives are defined by the following rules:

Rule 1: Each functional/process objective can correspond to one or more function/process.

$$\forall u, v \in F_{Id}, o \in O_f : (u, o) \wedge (v, o) \Rightarrow u = v \quad (8)$$
$$\forall u, v \in \mathit{power}(F_{Id}), o \in O_p : (u, o) \wedge (v, o) \Rightarrow u = v$$

Rule 2: Functional objectives and process objectives are a subset of the means objectives, with business process-modeling objectives at each level of the process hierarchy forming a level within the means objectives network.

$$O_f \cup O_p \subseteq O_m \quad (9)$$

The motivation for this rule is as follows:

- the definition of a means objectives discussed in Section 2.1 is consistent with the definition of a function (and correspondingly a process) discussed in Section 2.2 – both refer to the action required for a specific objective to be achieved;
- the levels within the network of functional objectives correspond to the levels within a process hierarchy; and
- the network of objectives developed in accordance with the rules discussed in Section 2.1 satisfies functional goals' requirements provided in Section 2.2.

Rules 3 and 4 below are derived by combining Keeney's definitions of means and fundamental objectives [12] p34 with the definitions and rules already described in this section. Rules 3 and 4 complete the set of rules describing the relationship between functional, process, means and fundamental objectives.

Rule 3: The set of functional objectives does not intersect with the set of fundamental objectives; similarly the set of process objectives does not intersect with the set of fundamental objectives.

$$Since \quad O_m \cap O_d = \emptyset \Rightarrow O_f \cap O_d = \emptyset \wedge O_p \cap O_d = \emptyset \tag{10}$$

Rule 4: The links $l_{m,d}$ between functional/process objectives and fundamental objectives are a subset of the set of links between means objectives and fundamental objectives. This rule allows process modeling objectives to be linked directly to the top level fundamental objectives as a link to a lower level fundamental objective is uniquely mapped to a top level fundamental objective through the fundamental objectives hierarchy.

$$l_{f,d} \cup l_{p,d} \subseteq l_{m,d}, \text{ where} \tag{11}$$
$$l_{f,d} := \left\{ \left(o_f, o_d\right) : o_f \in O_f, o_d \in O_d \wedge \tau_\mu\left(o_f, o_d\right) = \mathrm{Ofl} \right\}, \left| l_{f,d} \right| > 0$$
$$l_{p,d} := \left\{ \left(o_p, o_d\right) : o_p \in O_p, o_d \in O_d \wedge \tau_\mu\left(o_p, o_d\right) = \mathrm{Ofl} \right\}, \left| l_{p,d} \right| > 0$$

The objectives network satisfying the rules and definitions above is a "value-focused thinking" network applied directly to the e-EPC functional view. Functional goals are means objectives and strategic business goals are fundamental objectives. This model provides a broader decision context for the e-EPC and links means and fundamental objectives to processes and functions responsible for fulfilling them. The nature of the links is explored with Systems Dynamics tools in the next section.

4 Illustrative Example

In this section Human Resources (HR) context is used to illustrate the linkage model and to analyze relationships between different levels of objectives.

4.1 HR Context

HR have a critical role in positioning an enterprise to achieve its corporate objectives [8], [15], [31], [34]. Traditionally, HR departments provided leadership in the areas of staff development, remuneration, industrial relations and performance management. In recent years, the focus of the HR strategies in many organizations has evolved towards strategic workforce planning and providing line managers with tools and support for managing their staff [21]. Generally speaking, the focus of the HR information systems architecture and associated tools is on operational processes and functions such as administration of payroll and benefits, recruitment, and personnel management (e.g. [1]). Decision support tools mainly focus on decisions narrowly defined to fit within the specific techniques such as Markov chains (manpower planning), Linear and Integer programming (scheduling), Multiple Criteria Decision Analysis (selection), etc. [32]. The paths of the two methodologies rarely crossed due to the differences in paradigms and terminology used by the respective disciplines. The linkage model defined in the previous section can facilitate effective support of HR strategies by improving understanding of how process modeling and decision support objectives relate to each other.

4.2 HR Objectives within the "Value-Focused Thinking" Framework

HR literature provides a wealth of information about HR objectives, but these are rarely distinguished into fundamental and means objectives except to illustrate the application of "value-focused thinking" to a specific enterprise or decision context [6], [9], [12]. The aim of this example is to demonstrate the linkage model within the HR context rather than construct a universal objectives structure for HR functions. Nevertheless, we have confirmed with the Australian Government and private sector HR practitioners that the structure presented in this section is a fair representation of HR objectives in business and government within the Australian context.

To construct a single "value-focused" framework we have reformulated the objectives found in the HR literature (e.g. [3], [4], [5], [17], [21], [26], [28], [29], [30], [31]) to satisfy Keeney's definition of the objective, merging similar objectives under a more general objective and then classified the resulting objectives into a directed network in accordance with the principles outlined in Section 2.1. This resulted in three top-level fundamental objectives, twenty lower level fundamental objectives, six top-level means objectives and over 50 lower level means objectives. The top-level fundamental objectives were: "max value of HR services to meet organizational objectives" [5]; "max well-being of employees" [3]; and "max social legitimacy" [4].

Top level means objectives (e.g. [5], [17], [28]) were: 1) "ensure excellence in HR planning and research" (Planning); 2) "ensure excellence in staffing" (Staffing); 3) "ensure excellence in training and development" (Training); 4) "ensure excellence in industrial relations" (IR); 5) "ensure excellence in compensation" (Compensation); and 6) "ensure excellence and integration of technology of infrastructure relevant to HR functions" (Technology).

Complexity of lower levels of means objectives varies considerably from process to process reflecting different levels of complexity and detail that can arise when identifying objectives within an organization. For example, the Planning process means objectives consist of one lower level with five objectives, whereas the Training process objectives network (Figure 1) is considerably more complex.

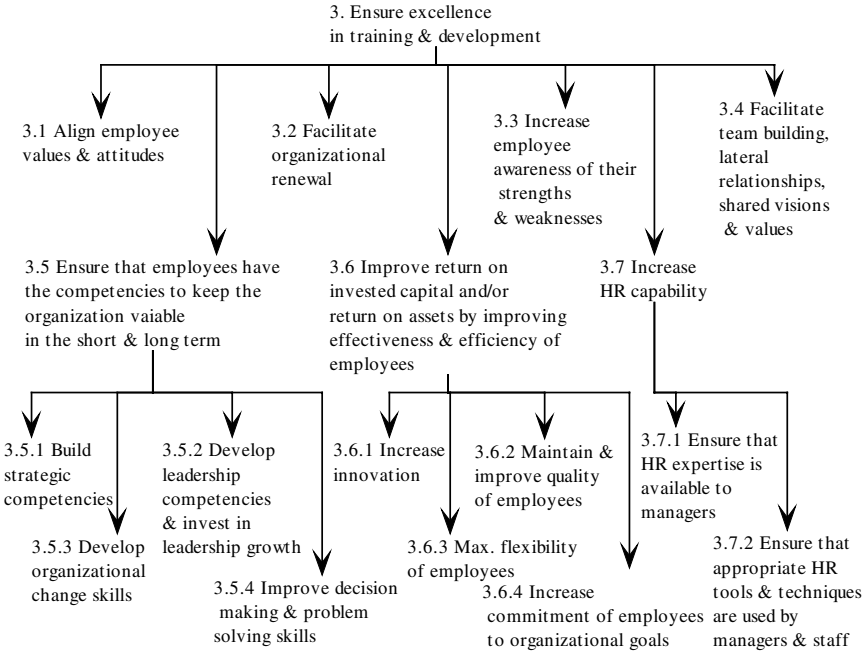

Fig. 1. The Training branch of the means objectives network (e.g. objectives 3.6, 3.6.3 [4], objective 3.6.2 [12], objectives 3.1, 3.3, 3.5.4, 3.6.1 [26], objective 3.4 [30], objectives 3.5, 3.6.4, 3.7, 3.7.1-2 [29], objectives 3.2, 3.5.1-3 [31]).

In HR literature strategic HR objectives are often supported by a description of HR processes, functions and decisions aimed at achieving these objectives (e.g. [3], [17]). Likewise in the constructed network, the top level of the HR means objectives corresponds to the six core HR processes used to achieve the fundamental HR objectives. The next level of means objectives appears to relate to both specific sub-processes and specific goals within the main process (see Figure 1).

4.3 HR Process Model

For the purposes of illustrating HR processes, standard HR texts (e.g. [5], [17], [28]) were used as a basis for reference models for the core HR processes identified in the previous section.

The processes were modeled using event-driven process chain methodology [14], [20], [23], [24], [25]. An example of the process model for the Training process ([5]

ch. 12, [17] ch. 11, [28] ch. 9) is provided in Figure 2. The functional goals were based on texts used to construct corresponding processes as well as the network of HR objectives constructed in the previous Section. For example, Milkovich [17] and other HR authors indicate that evaluation is required in order to "ensure" that HR process objectives are met, therefore "evaluate training outcomes" function and corresponding objective was included in the EPC to provide a complete process description and to enable objective 3.5 in Figure 1 to be met.

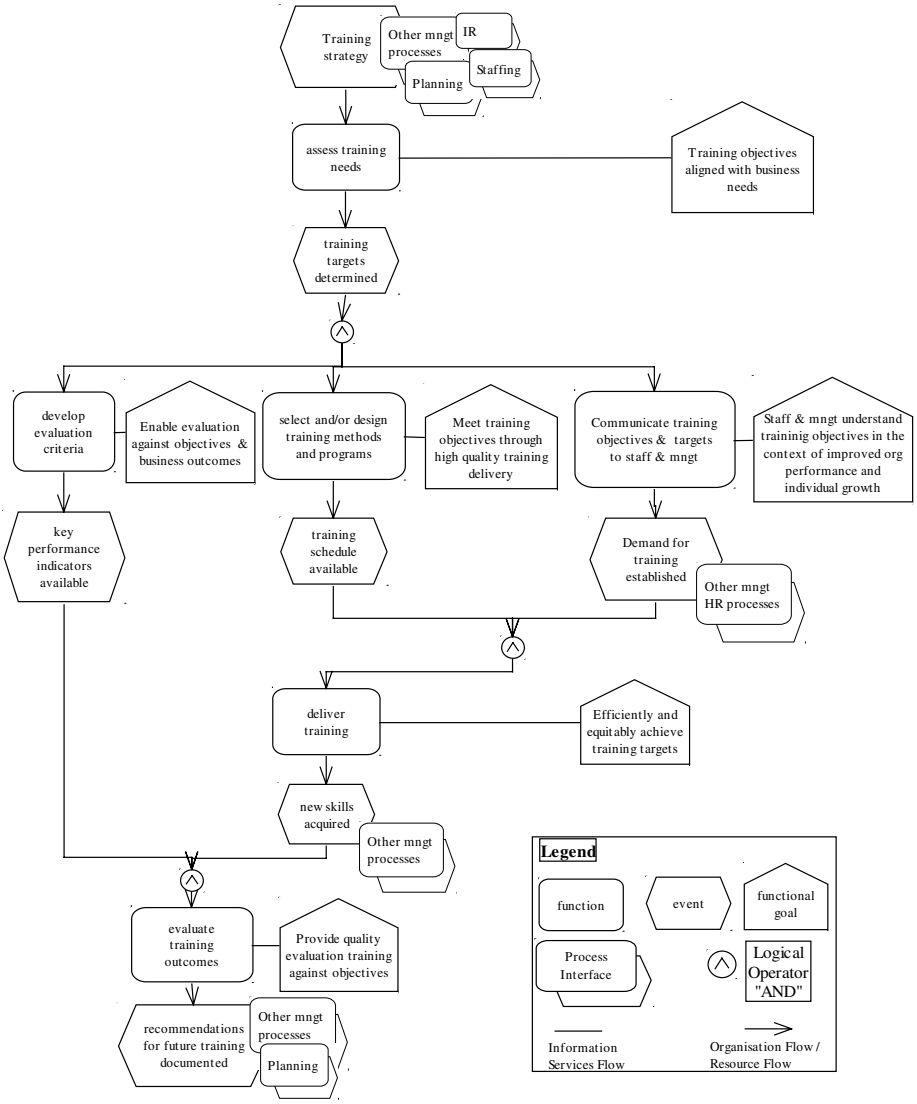

Fig. 2. Business process model (using e-EPC) for the Training process.

In addition to functional goals, three goals were identified for the overall Training process: "use resources efficiently and effectively"; "develop competencies"; and "align behaviors and values".

4.4 Relationship between Different Levels of Objectives

In order to construct a single objectives hierarchy consistent with the linkage model presented in Section 3, the relationship between objectives in Section 4.2 and 4.3 was analyzed. It was found, consistent with the linkage model, that process and functional objectives represented different levels of means objectives. Furthermore, functional objectives presented the means of achieving some process objectives and both functional and process objectives presented the means of achieving the fundamental objectives.

Consistent with the linkage model that stipulates that process and functional objectives are a subset of the means objectives it was also found that not all means objectives identified in Section 4.2 were included in process or functional objectives identified in Section 4.3. Many means objectives illustrated in Figure 1 represented lower level functional objectives (e.g. objectives relating to specific training skills such as "decision and problem solving skills" objective in Figure 1) that were not included at the level of the process model illustrated in Section 4.3. Others were found not to be relevant to this specific process in their initial form (e.g. "increase HR capability" objective in Figure 1). As a result of this analysis it became apparent that linkage model should be extended in the future to include rules that will aid synchronization of the levels at which objectives network and process model are constructed.

Due to the difference in the levels between the objectives network (Figure 1) and the process model (Figure 2), it initially appeared that the linkage model rules were not fully satisfied as a number of process and functional objectives described in Section 4.3 were not part of the "value-focused thinking" structure (e.g. "Provide quality evaluation training against objectives" is not included in Figure 1). Upon closer examination, these objectives satisfied the definition of the means objectives and, therefore, were added to the "value-focused thinking" structure to satisfy the linkage model requirements. Reconciling the two hierarchies resulted in revisions to the process model and objectives network initially developed.

Further insight into the relationship between objectives could be gained through the use of System Dynamics, and in particular, the stock and flow diagram (SFD). The stock and flow diagram (SFD) is a well established methodology in the business dynamics tool kit that uses mental representation of a business to build a quantitative simulation model of causal relationships between key business quantities and feedback mechanisms within the business [27]. The SFDs and corresponding simulation models provide a more complete representation of the system than the objectives network since they capture the states (stocks) of the system, the rates (flows) at which these system states change and causal and feedback mechanisms [27].

For example, the relationship between the process objective of "decreasing the gap between existing and desired competencies" and fundamental objective of "maximiz-

ing well-being of employees" is modeled as a reinforcing feedback loop in an SFD: increasing gap between existing and desired competencies would decrease the well-being of employees, decreasing well-being is likely to increase the number of unplanned staff resignations that will in turn further increase the size of the gap.

Through modeling causal loop links between objectives and the nature of these links (i.e. reinforcing or balancing), an SFD model facilitates identification of counter-intuitive or hidden causal relationships. Furthermore, by translating organizational objectives into a set of Key Performance Indicators (KPIs) simulation can be used to model the behavior of the system that would lead to better understanding of the processes and provide opportunities to refine both process models and objectives network [19].

The final objectives network presented in Figures 3 was the result of a number of iterations required to incorporate feedback from each of the modeling techniques described in this paper and contains only those means objectives that are relevant to the level of the process model illustrated in Figure 2.

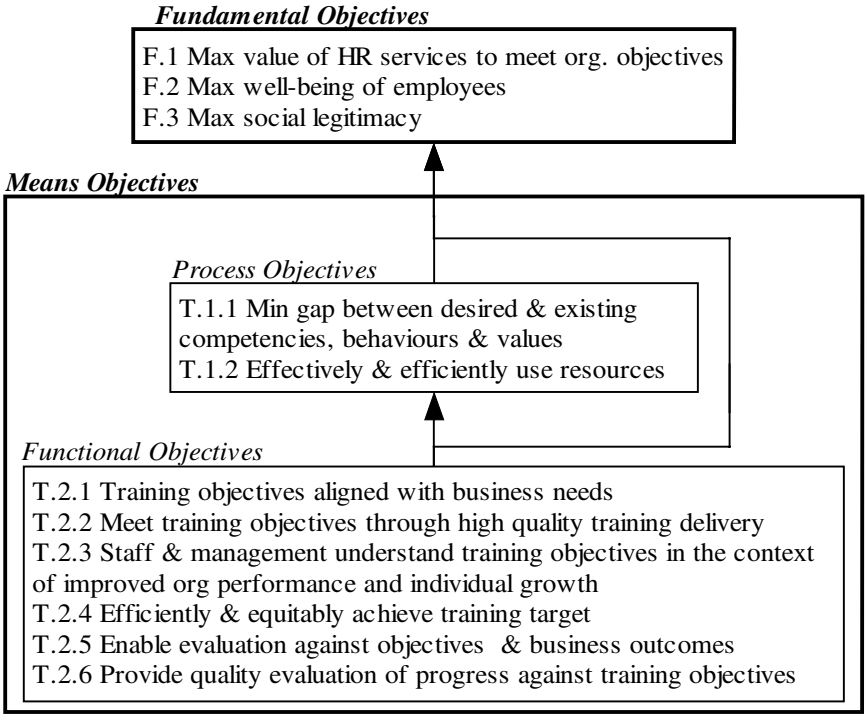

Fig. 3. Final objectives network for the Training process business model.

5 Implementation Guidelines

For the formalism presented in Section 3 to be of value to business engineering practitioners, it needs to be accompanied by an implementation model. Based on the experience described in the previous section we recommend a four-stage iterative implementation process (Figure 5) with each stage focusing on a specific model while allowing feedback between stages.

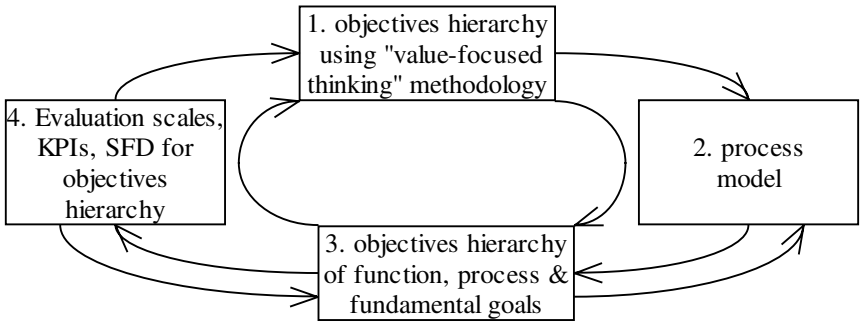

Fig. 4. Iterative process for the implementation of the linkage model.

In Stage 1, a business analyst (or a team of analysts depending on the scope and the time frame for the project) in consultation with the key decision makers needs to produce an objectives network using "value-focused thinking" methodology. The top level means objectives from this network guides the next stage by identifying key processes required to achieve the fundamental objectives.

Through observing business processes or using a reference model in the context of Business Process Re-eneginering, the analyst constructs process models in Stage 2. The process model should have means objectives assigned to each function taking into account objectives already specified in Stage 1.

Using information from Stage 2, Stage 3 assigns means objectives to higher-level processes and combines them with functional objectives to form a new objective network. Models in Step 1 and Step 2 may need to be revised to reflect linkage model rules. Means objectives that do not contribute to the objective network in this stage should remain in the Stage 1 model as they may be required for a lower level process model or may be relevant for other purposes (e.g. decision making).

In Stage 4, the analyst should first identify a set of KPIs corresponding to the objectives defined in Stage 3 and then use it to construct an SFD. Additional information gained in Stage 4 contributes to a review of all models.

A final network of objectives incorporating "value-focused thinking" and process model objectives is available once all stages are complete, consistent and comply with the linkage model rules.

6 Discussion

The proposed linkage model has major implications on the interpretation of the role of processes and functions within an enterprise. The model re-emphasizes that the role of the business process model is to describe *how* the fundamental objectives of an enterprise (reflecting what is important for the organization) are to be achieved while emphasizing that the process and functional objectives alone do not represent the fundamental objectives of the enterprise. This differentiation is important since it ensures via the links between process and fundamental objectives that the focus of the process is on actions set within a wider organizational context.

Defining process and functional objectives as means objectives also has the benefit of converting the informal link between processes and enterprise objectives proposed by some enterprise modeling researchers (e.g. [35]) into a structured process that reflects the fundamental values of the enterprise for the identification of process modeling objectives. Application of this process results in process and functional goals being clearly linked to the fundamental organizational and decision-making objectives.

The model also has important implications on the interpretation of the means objectives network within the "value-focused thinking" methodology. In particular, the structure of the network becomes considerably more focused and transparent as each level of the means objective network is reflecting a corresponding level within the functional network in the process model. The lowest level means objectives correspond to the elementary functional objectives and the top-level means objectives correspond to the top level processes in the model. Means objectives that are not in the process model are easily identifiable facilitating early identification of appropriate action (e.g. decision or additional processes) necessary to fulfill these objectives.

Additional insight is reached as a result of quantitative modeling of the relationship between objectives using decision analysis tools such as the stock and flow diagram and corresponding simulation model. For example, causal links between specific objectives both within and across levels of the network become clearly evident in an SFD model. On the whole, application of the linkage model highlighted "silent" features of the two modeling approaches and was able to provide a path towards a more comprehensive enterprise model.

7 Summary and Conclusions

Discussion in this paper highlights the benefits of bridging the gap between process and decision modeling by linking objectives generated by these methods. The formalized linkage model facilitates the integration of strategic, decision and process objectives within a single framework. In essence, the model defines process and functional objectives as a subset of means objectives and proposes the rules for structuring an integrated network. Application of the stock and flow diagram provides further insights into the relationship between objectives. Illustration of the model in the HR

context results in a proposal for a four-stage iterative model to be used as a step-by-step guide to identification of objectives within a business-modeling context.

The model would further benefit from application to real-life organizations, extension of linkage rules to guide synchronized decomposition (and expansion) of the objectives hierarchies and process models, and further exploration of quantitative relationships between objectives using decision analysis tools.

Acknowledgements. The authors would like to thankfully acknowledge valuable comments of three anonymous referees.

References

1. Anderson, L.: Understanding Peoplesoft 8. Sybex SanFrancisco (2001)
2. Blain, J., Dodd, B.: Administering SAP R/3: the HR-Human Resources Module. QUE Indianapolis (1999)
3. Boxall, P. F.: Strategic Human Resource Management: Beginnings of a New Theoretical Sophistication? Human Resource Management Journal, Vol. 2, No. 3 (1992) 61–79
4. Boxall, P. F.: Human Resource Strategy and Industry-Based Competition: a Conceptual Framework and Agenda for Theoretical Development. In Ferris G. F. (ed.): Research in Personnel and Human Resource Management, Supplement 4 JAI Press Stamford London (1999) 259–281
5. Cascio, W. F., Awad, E. M.: Human Resource Management: an Information Systems Approach. Reston Pub. Co. Virginia (1981)
6. Clemen, R. T., Reilly, T.: Making Hard Decisions with DecisionTools. 2nd rev. edn. Duxbury, USA (2001)
7. Davis, R.: Business Process Modelling with ARIS: a Practical Guide. Springer-Verlag, London Berlin Heidelberg (2001)
8. Gratton, L., Hope-Hailey V., Stiles P., Truss C.: Linking Individual Performance to Business Strategy: the People Process Model. Human Resource Management, Vol. 38, No. 1 (1999) 17–31
9. Gregory, R., Keeney, R. L.: Creating Policy Alternatives Using Stakeholder Values. Management Science Vol. 40 No. 8 (1994) 1035–1048
10. IDS Scheer: ARIS Methods Manual. Version 5 IDS Scheer (2000)
11. Keeney, R. L.: Value-Focused Thinking: A Path to Creative Decision Making. Harvard University Press, Cambridge, Mass : Harvard University Press (1992)
12. Keeney, R. L.: Creativity in Decision Making with Value-Focused Thinking. Sloan Management Review Summer (1994) 33–41
13. Keeney, R. L.: The Value of Internet Commerce to the Customer. Management Science Vol. 45 No. 4 (1999) 533–542
14. Keller, G., Teufel, T.: SAP R/3 Process – Oriented Implementation: Iterative Process Prototyping. Addison Wesley Longman, Harlow, England (1998)
15. Khoong, C. M.: An Integrated System Framework and Analysis Methodology for Manpower Planning. International Journal of Manpower Vol. 17 No. 1 (1996) 26–46
16. Martin, N., Gregor, S.: Enterprise Architecture and Information Systems Alignment: Policy, Research and Future Implications. In Wenn, A., McGrath, M., Burstein, F.: Proceedings of the 13th Australasian Conference on Information Systems. Vol. 3 (2002) 1057–1068

17. Milkovich, G., Boudreau, J. W.: Human Resource Management. 6th edn. Irwin Chicago (1991)
18. Moussa, N., Moussa, V.: Constructing a Multicriteria Hierarchical Evaluation Model Using an Aggregation-Disaggregation Approach. Proceedings of the 5th DSI-99 Conference, Athens July 4–7 (1999) 1403–1409
19. Neiger, D., Churilov, L.: Towards Decision-Enabled Business Process Modelling Tools: from e-EPC to de-EPC. In Wenn, A., McGrath, M., Burstein, F.: Proceedings of the 13th Australasian Conference on Information Systems. Vol. 1 (2002) 151–162
20. Nuttgens, M., Field, T., Zimmerman, V.: Business Process Modeling with EPC and UML: Transformation of Integration? In Schader, M., Korthaus, A. (eds.): The Unified Modelling Language – Technical Aspects and Applications, Proceedings (Mannheim, October 1997), Heidelberg (1998) 250–261
21. Rahman bin Idris, A., Eldridge, D.: Reconceptualising Human Resource Planning in Response to Institutional Change. International Journal of Manpower Vol. 19 No. 5 (1998) 343–357
22. Rockart, J. F.: Chief Executives Define Their Own Data Needs. Harvard Business Review March/April (1979) 81–97
23. Scheer, A.-W.: Business process engineering: reference models for industrial enterprises. Study edn. Springer-Verlag, Berlin, Heidelberg, New York (1998)
24. Scheer, A.-W.: ARIS – Business Process Frameworks. 3rd edn. Springer-Verlag, Berlin Heidelberg (1999)
25. Scheer, A.-W.: ARIS – Business Process Modeling. 3rd edn. Springer-Verlag, Berlin Heidelberg (2000)
26. Schuler, R. S., Walker, J. W. (eds.): Managing HR in the Information Age. Bureau of National Affairs Washington (1991)
27. Sterman, J. D.: Business Dynamics: Systems Thinking and Modelling for a Complex World. The McGraw-Hill Companies, USA (2000)
28. Stone, R. J.: Human Resource Management. 3rd edn. Jacaranda Wiley, Queensland (1998)
29. Treasury Board of Canada Secretariat: Human Resources Management Framework – a Reference Tool for Managers. 2001 ed. Minister of Public Works and Government Services Canada (2001)
http://www.tbs-sct.gc.ca/hr_connexions_rh/sigs/Framework/FRAME_e.html (last accessed 24/12/02)
30. Walker, J. W.: Human Resource Planning, 1990's Style. Human Resource Planning Vol. 13 No. 4 (1990) 229–240
31. Walker, G., MacDonald, J. R.: Designing and Implementing an HR Scorecard. Human Resource Management Vol. 40, No. 4 (2001) 365–377
32. Winston, W. L.: Operations Research: Applications and Algorithms. Wadsworth USA (1994)
33. Winthrop, M. F.: C2ISR Low Demand/High Density (LD/HD) Aircraft Investment and Organizational Solutions. (2000)
www.mors.org/education_colloquium/EC2000/presentations/winthrop.ppt (last accessed 18/12/02)
34. Zeffane, R., Mayo, G.: Planning for Human Resources in the 1990s: Development of an Operational Model. International Journal of Manpower Vol. 15 No. 6 (1994) 36–56
35. Yu, E. S. K., Mylopoulos, J., Lesperance, Y.: AI Models for Business Process Reengineering. IEEE Expert August (1996) 16–23

Use Cases as Workflows

Michel Chaudron, Kees van Hee, and Lou Somers

Eindhoven University of Technology, Dept. Math. & Comp.Science, P.O. Box 513,
NL-5600 MB Eindhoven, The Netherlands
{m.r.v.chaudron, k.m.v.hee, l.j.a.m.somers}@tue.nl

Abstract. In requirements engineering we have to discover the user requirements and then we have to transform them into precise system specifications. There are two essential aspects to be modeled: the data aspect and the process aspect of the system. There are many techniques available to describe these aspects but it is always difficult to integrate these views in a consistent way. Last decade two techniques are used frequently in requirements engineering: use cases and workflow models. We show that these techniques can be integrated in a natural way, using the framework of colored Petri nets. We only sketch the underlying formal framework and focus on the practical application of the approach by a case study.

1 Introduction

Requirements engineering is a distinguished field in software engineering since many years (see e.g. [16]). There are two famous problems: one is to bridge the gap between informal requirements and formal specifications, the other one is to integrate models that describe different aspects of a system. Informal requirements are important for the principals and the potential users of a system, formal specifications are essential for the software constructors. The last decade *use cases* are used more and more as a way to describe requirements. It seems that non-experts understand them better than for instance data models and dataflow diagrams. There is no formal definition of a use case that is accepted by a large community of software engineers. It is generally understood that a use case describes a "piece of functionality" of a system from the viewpoint of an actor who will work with the system. A use case combines in fact a data and a process perspective. In the early stages of system development, where the concept has to be "sold" to the principals and potential users it might be an advantage that use cases do not have a formal definition: this gives us some freedom to apply the concept and to interpret a specific use case. However, for the following phases this lack of precise semantics is a source of problems for the software engineers. Therefore, it is essential to have formal semantics for use cases. We use workflow theory for this purpose.

In the field of business process design the use of modeling techniques with a precise semantics has proven to be very useful: errors and missing parts become clear in the early stages and it saves work later on. The availability of techniques for verifica-

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 88–103, 2003.
Springer-Verlag Berlin Heidelberg 2003

tion of models turned out to be very valuable in practice [2]. Another term for business process is workflow. There are several formalisms to model workflows, one very successful one is to describe workflows as a special class of Petri nets, the so-called *workflow nets* [4].

In this document we apply results of workflow theory to model use cases. In fact, we consider use cases as workflows and we show how the field of requirements engineering can profit from the results of the field of business process modeling. A nice coincidence is that the term "case" occurs in workflow theory for "job" to be handled and so a workflow can be seen as a *case type*. Therefore, a use case is modeled as a case type in the sense of workflows.

The second problem is the integration of models that describe different aspects of systems. There are many articles about this subject. The use of colored Petri nets as a framework for integration has proven to be successful (see for instance [9] or [12]). Here we use the same approach, although we simply say: use cases could be modeled as workflow nets.

The remainder of this paper is organized as follows. In Section 2 we consider the place of requirements in the life cycle of a system and the steps to take to describe the functional requirements for a software engineering project. So non-functional requirements (availability, performance, adaptability, portability and many other "abilities") are not covered in this paper. In Section 3 we consider the modeling concepts and in Section 4 we summarize the most important construction techniques. In Section 5 the main course is served: a case study where we show how the proposed approach should work.

2 Requirements Engineering in the Lifecycle

There are many standards for phasing the lifecycle of a software system. Rational Unified Process (RUP [13]) is one newest leaves of this tree. We have chosen here the standard [17] of the European Space Agency (ESA). The reason for this is that it is a well-documented standard, that the standard is used frequently in practice, and that last but not least we adopted this standard some years ago for our students in computer science. The choice of a standard for our purpose is not essential: they all distinguish requirements in some form.

The ESA standard distinguishes the following phases in the lifecycle of a software system: user requirements, software requirements, architectural design, detailed design and production, transfer, operations and maintenance.

Here, the first two phases refer to requirements. We refer to these phases together as the "requirements phase" with two sub-phases: "user requirements" and "software specification" because here the main course is the formal specification of the system.

Each phase has some deliverables as output: the product documents that determine certain aspects of the system. We do not consider the strategy to perform the phases, so our contribution is applicable to *sequential* (or waterfall), *iterative*, *incremental*, or *time boxing* strategies. In this paper we will concentrate on the production of software requirements and because this phase is in between the user requirements and archi-

tectural design, we touch these topics as well. The user requirements have a certain overlap with the software requirements, but they are written in the language of the user. The software requirements are meant for software engineers and they contain much more details. In our approach we promote the use of formal methods in the early stages because these methods encourage us to be precise and enable us to use verification methods. In this way we are able to discover inconsistencies and errors in a very early stage, which will pay off in later phases: the later an error is found the more expensive it is to correct it.

In the software requirements phase we concentrate on the functionality of the system to be made. We encourage having a complete *logical* model of the software in this phase. (We use here a pragmatic definition of the term "complete": a model is complete if it contains enough details to build or generate a computer model of the system.) A logical model of a system is often considered as a formal specification of a system. In the architectural design this logical model is translated into the specifications for the *real* system. In both phases we distinguish "components". In the software requirements the system is one logical component or it may be refined into several communicating logical components. In the architectural design the components are physical components: pieces of software, implemented on hardware. It is necessary to map the logical components to the physical ones. Sometimes it is possible to have a one-to-one mapping. However, in many cases one logical component is distributed over several physical components. It also occurs that several logical components are realized by one physical one.

The activities to be performed in the first two phases are as follows. Some terms are not explained here but they will be in the next sections. In the user requirements phase one has to:

1. Identify the *stakeholders*: all relevant roles that have some interest in the system.
2. Identify the *actors*: all stakeholders and other systems that interact with the system.
3. Describe (informally) the *use cases*: the logical "pieces of functionality".
4. Describe the *non-functional* requirements like performance or portability. This includes development constraints like the development environment or the execution platform.

In the software requirements phase one has to:

5. Transform the use cases into *workflows* (WF-nets).
6. Make a *class* model.
7. Describe class *lifecycles* (workflows).
8. Describe the *interactions* between all workflows.
9. Connect the workflow *transitions* and the *methods* in the class diagrams.
10. Perform logical system *decomposition* (if necessary).

Steps 5 and 6 may be performed in reversed order. In fact, all these tasks are performed iteratively, because in tasks 8, 9, and 10 we may discover inconsistencies that have to be repaired by redoing earlier tasks. In the software requirements phase also requirements in the form of necessary or desired properties may be formulated in terms of logic. In addition, non-functional requirements such as timeliness may be formulated in a formal way in this phase. However, we concentrate on the functional requirements here. The results of the first steps are described in the User Requirements Document (URD), the others in the Software Requirements Document (SRD). The URD roughly describes the problem the stakeholders need to address, whereas the SRD describes the specification of the proposed solution.

3 Modeling Framework

In this section we introduce the modeling concepts that are suitable to perform tasks 5 up to 10, i.e. we consider the formalisms with which we model the various aspects of a system. It is popular to use the modeling techniques of the Unified Modeling Language (UML) (see [7] for a concise introduction). This framework consists of many useful modeling techniques. In fact, we only use uses cases, class models and (a variant of) activity diagrams from UML. This does not imply that we abandon the others, but we focus here on the integration of only a few of the modeling techniques.

The things we have to model of a system are in fact the *state space* and the *events* that may change the states of the system. A state of system is defined by a set of *data objects* modeled by means of a *data* model. For events we use a *process* model. (At the end of this section we will establish the relationships between the events and data objects.)

Persistent data objects are modeled by the *class* model of UML, i.e. data objects that remain in the system if there are no events. The UML class model also allows us to define operations on the data, the *methods*. The *volatile* data that are produced and consumed during the events can be modeled with a standard *type system* or (if it is complex data) with a language like XML.

For the *process modeling* we use (high level) *Petri nets*. (For an introduction, standard terminology and many important properties of Petri nets we refer to [15] and [6].) Events are modeled by (the firing of) *transitions* and states are modeled by the *marking* of places. Petri nets are very close to UML *activity diagrams,* but have better defined semantics and many possibilities to verify behavioral properties. An alternative could be to use a *process algebra*.

The *process models* occur in two forms in our specifications: as use cases and as lifecycles for object classes. We require that these processes are a special kind of Petri nets: *workflow nets* [4]**.** Workflow nets have one initial and one final place and each node of the net lies on a path from the initial to the final place.

An additional requirement is that the workflow nets are *sound* (c.f. [1]), i.e. each reachable state (or marking) of the net lies on a path from the initial to the final state in the state space, where the initial state has only one token in the initial place and the

final state has only one token in the final place. The reason for requiring that all use cases and lifecycles are sound workflow nets is that this guarantees that all use cases and lifecycles are proper *transactions* of the system: with a start and an ensured end.

The *transitions* in a use case model are *activities* of the actors that are involved in the system: users or other systems. Transitions can also be *autonomous*, which means that they are executed by the system itself as soon as the transition is enabled. In many situations the transitions are *triggered* by users in a user interface.

Tokens in the workflow nets that model use cases, represent the *case* that is handled. If there is concurrent behavior possible, then a case is represented by more than one token. Then we assume that each token has a (case) identity and that transitions, that consume more than one token at the same time, only consume and produce tokens with the same identity. Besides a case identity, a token may carry some message (volatile) data. Therefore, the tokens may have values, which means that we work with colored Petri nets [11].

We often have some global variables that are used in transitions. They are modeled by *global stores*, i.e. places that always contain one token and that are connected to many transitions by a read and write arc. (We normally do not draw these arcs to global stores in the diagrams.) The instances of a class model may be stored in a global store. Therefore, we may associate sub-class models to global stores. In fact, different global stores may have the same sub-class models.

Since we model with colored Petri (or workflow) nets, we obtain the relationship between data objects and events in a natural way: the states are modeled by the places marked with colored tokens (which are the data objects) and the events are modeled by transitions that change the states by consuming and producing colored tokens.

The last concept we need is the t-workflow. This is a Petri net with an initial and final transition instead of initial and final place. It is easy to transform one into the other: put a place (transition) in front of the initial transition (place) and a place (transition) after the final transition (place) and one obtains the other form of workflow. In addition, the concept of soundness can be translated for t-workflows: a t-workflow is sound if and only if its transformation is a sound workflow net.

4 Construction Techniques

There are several ways to build Petri nets in a structured way (see e.g. [8]). Here we do not present an exhaustive list of construction techniques; we only present some important techniques for the construction of workflow nets.

For workflow nets, it is essential that the workflows we model are sound. Non-sound workflows usually have serious modeling errors. Soundness is not the only property worthwhile to be verified, but it is the most important domain-independent "sanity check" for models. There are efficient techniques for verifying the soundness of workflows [18]. So one can model a workflow and check afterwards whether it is sound or not automatically. We have good experience with an approach that guarantees soundness by the way we construct the models: "soundness by construction".

Essentially, there are two approaches to construct workflow nets in a systematic way:

1. *Top down*: by stepwise refinement of places and transitions in a given start net.
2. *Bottom up*: by connecting existing component nets.

In practice, these approaches may be combined.

For the top down approach we have only one transformation rule: *replacement*. We start with a given set of basic workflow nets or *workflow patterns* for which we have a proof of soundness. Then we refine or replace a place by a sound workflow net or a transition by sound t-workflows. (There are some difficulties if a sub-workflow can be triggered two or more times concurrently, cf [10]). In most cases the result is a sound workflow again.

In the bottom up approach we also need only one basic transformation rule: *fusion* of places or transitions, i.e. two places (transitions) are "glued" together to become one place (transition). We have some standard constructions, sometimes using some auxiliary building blocks (also Petri nets) to assemble workflow nets: *sequential, alternative, parallel* and *iterative* composition. In sequential composition we fuse the final place (transition) of the first (t-)workflow net with the initial place (transition) of the second (t-)workflow net. In alternative composition, we fuse the initial places and the final places of two workflow nets. For the alternative composition of t-workflows, we need some auxiliary blocks: an or-split and an or-join (see Fig. 1). The parallel composition of t-workflows is simply the fusion of the initial and final transitions of the two t-workflows. For the parallel composition of ordinary workflow we need two auxiliary blocks: an and-join and an and-split (see Fig. 1). For the iterative composition we need another auxiliary building brick: a sequence of two places with a transition in the middle. The first place is fused with the final place of the workflow net and the last place of the building block with the first place of the workflow net. To make a correct workflow net of this we use this auxiliary block twice: one in front of the iterated net and one at the end. In case of a t-workflow, we use a similar construction.

There is one other important standard construction: *asynchronous coupling* between two workflow nets. In this case, we use an auxiliary building block, which is a sequence of two transitions with a place in the middle. We fuse the first transition with some transition of one of the two workflow nets and the last transition with some transition of the other workflow net. Note that in this way we may loose the workflow structure and even if we have a connected them such that the overall net is again a workflow net, then it is not sure that it is sound. However, there are some constructions that guarantee soundness (see [5]). Besides the asynchronous coupling, we have the *synchronous coupling*: where we just fuse transitions of two workflow nets, not being the initial or final ones.

Synchronous and asynchronous couplings are used frequently when we interconnect use cases and lifecycles. Synchronous coupling is applied if two events in different use cases are in fact the same. This construction suffers from the same risks as the asynchronous one concerning soundness. Asynchronous coupling is used if two workflows do not have overlapping transitions but still need some coordination: a transition of one workflow may only execute if a transition of another workflow has

executed before. In a synchronous coupling transitions have to execute simultaneously, whereas in an asynchronous coupling transitions have to execute in some order. In step 8 (see also section 5.4) of the requirements phase we apply these couplings. There we use a notation technique where we list the transitions per use case and lifecycle and where we relate them (by an arc) to transitions of other use cases or lifecycles. The relationship has a direction if one of the transitions is taking the initiative: one transition is *triggering* the other. If each of the related transitions may take the initiative, or if they have to execute simultaneously there is un undirected relationship. The choice for synchronous or asynchronous coupling may be delayed, sometimes even to step 10.

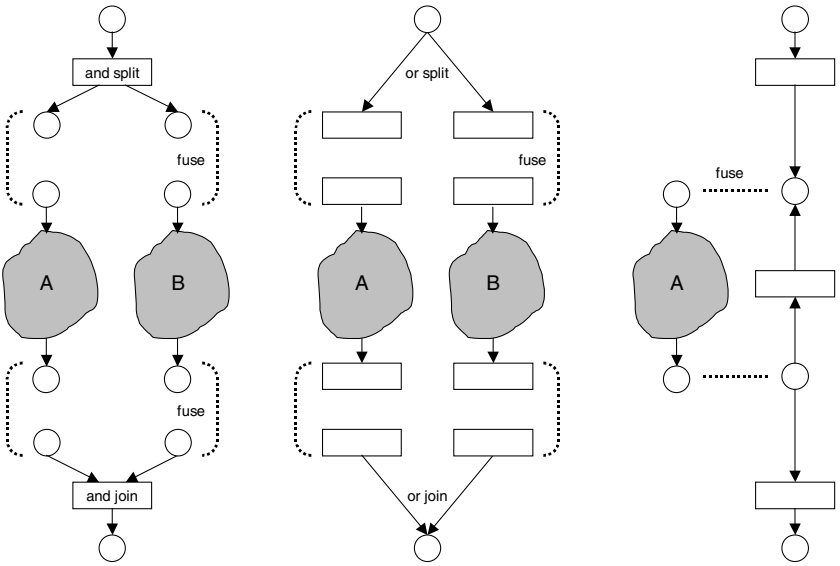

Fig. 1. Two constructions to compose workflows A and B. Iteration of workflow A

There are two other constructions we like to mention: the use of *global stores* and of *global transitions*. As mentioned before, a global store is a place that always contains one token and that is implicit (i.e. without drawing arcs) connected to a set of transitions with one consume and one produce arc. So the global stores do not influence the process flow of our workflow nets, but they are used to store variables that are shared by different transitions. Typically they are used to store a set of objects (the instance) of one class. Global transitions are implicit connected to a set of places (called "superplace") and if they are enabled by normal places, they consume all tokens available in the superplace. This can be used to enforce soundness of a workflow net and it is used frequently to model exception handling: if some event occurs we have to cancel the whole transaction. (Note that it is not always possible to simulate the behavior of a global transition without using inhibitor arcs.)

5 Case Study: The Web Shop

To illustrate the concepts treated in the previous section, we will develop the requirements for a web shop. For a part, a web shop is an ordinary shop, with items that are stored in warehouses, can be purchased by customers, are paid for, and are shipped. We will use the shopping cart metaphor, where a customer puts the products he wants to buy in a cart. In our web shop, a customer buys products that can be configured according to a certain model. So a model is a type of product. Computers or cars are typical examples of products in our web shop, but also holiday trips where the configuration is in fact the trip design. In this case study we focus on configurable physical products.

The actors are the persons, organizations, or systems that interact with the system we are going to build. In the web shop example, these might be a customer (browsing, buying, monitoring, feedback), inventory control (back ordering, shipping), the system administrator (back-up, upgrades), controlling (billing), marketing (pricing, product profiling, changing product portfolio), or design (page layout, styling).

5.1 Use Case Workflows

A use case corresponds to a task the system has to fulfill. Each use case involves a number of actors. We will show the workflows of a number of customer related use cases for the web shop and the rationale for the choices that have been made.

Use case "customer walks shop". As a first example, we will consider the use case "customer walks shop". The corresponding workflow is displayed in Fig. 2. Here the customer browses through the different models, configures products, adds products to his shopping cart, and possibly removes products from his shopping cart. Note that we might have modeled these steps also as four different use cases.

In the workflow definition of Fig. 2 we have constrained the cart manipulation: once the customer has selected a model, he has to put a product belonging to this model in his shopping cart, or he has to deselect it before he can view the contents of his shopping cart. A possible solution would be to define two screens allowing the customer to do both at the same time: we introduce parallel workflows. This is shown in the workflow at the left-hand side of Fig. 3. The matching of the input tokens by the exit transition is done on the case identity of the token.

Another issue is the fact that it would be more realistic to allow an exit from every state (since the customer may leave the site at any time). This is shown in the workflow at the right hand side of Fig. 3. Note that the exit may be an explicit user action, or might also be triggered by a timer event. It would be best to indicate explicitly who initiates each action (which actor or the system itself).

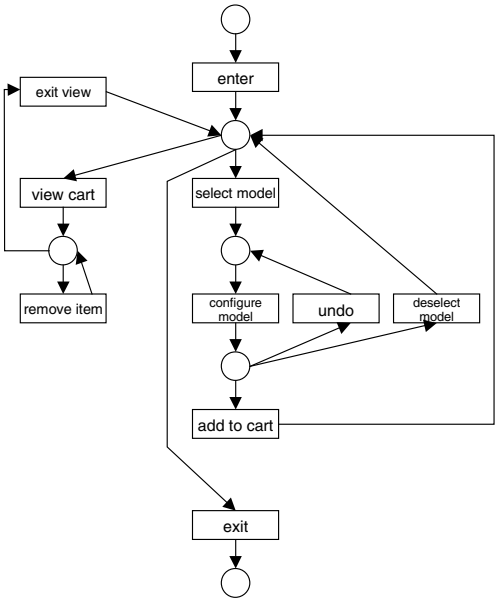

Fig. 2. First attempt to model the workflow of the use case "customer walks shop"

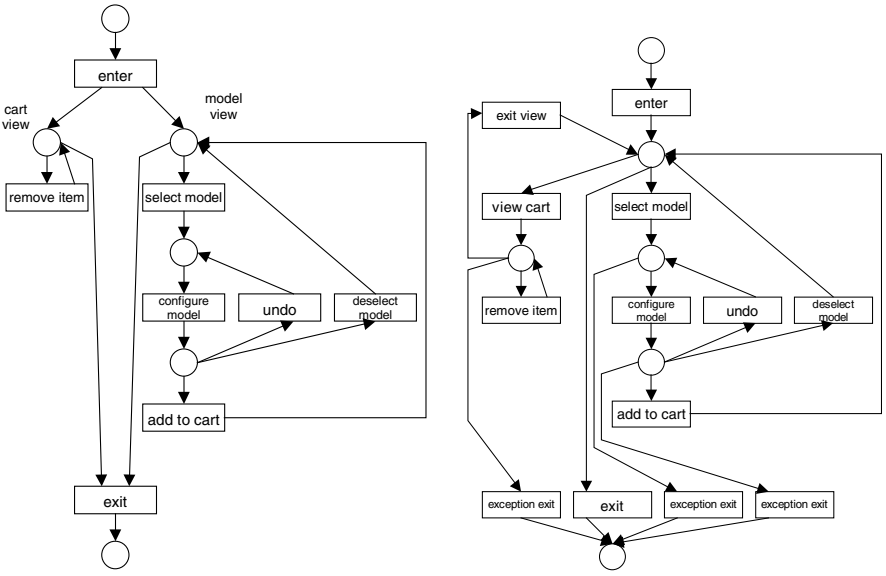

Fig. 3. Two alternatives for the use case "customer walks shop". The first one allows the customer to configure products and to view and manipulate the contents of his shopping cart at the same time. The other one allows an exit at any time

Use case "customer buys products". Another example is the workflow corresponding to the use case "customer buys products". Here the customer has a non-empty shopping cart and wants to buy these items. Two equivalent models are shown in Fig. 4. We have used a "super place" to model the exceptional exit flows.

For each choice of payment method (inter bank, credit card, cash on delivery, or paycheck), some customer solvability check has to be applied. The inner workings of these workflows may be different and will show up once we try to detail the transition "check customer solvability". We might also add activities to check the existence of the customer data and to add newly filled in data into the customer administration.

Use cases "back office" and "handling". The "back office" use case controls the work after the customer has agreed upon buying a product. The workflow is shown in Fig. 5. Only for pay on delivery, we have modeled the possibility that the customer does not pay.

We will use a simple version of the use case "handling" in which the ordered products are assembled and shipped. Note that for such workflows (like the shipping operation) some standard patterns exist in the literature.

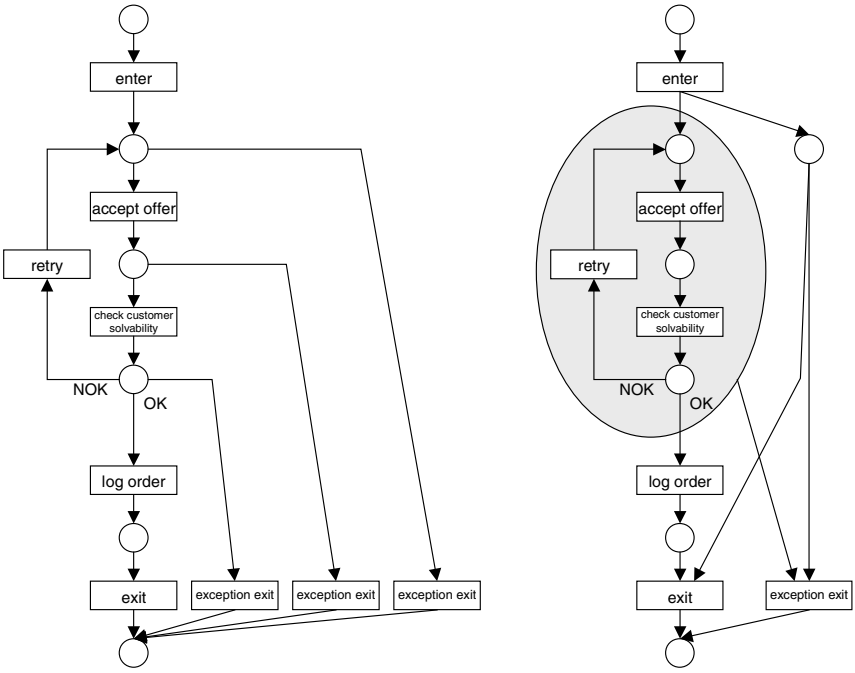

Fig. 4. Use case "customer buys products", variants without and with a superplace and a global transition ("exception exit")

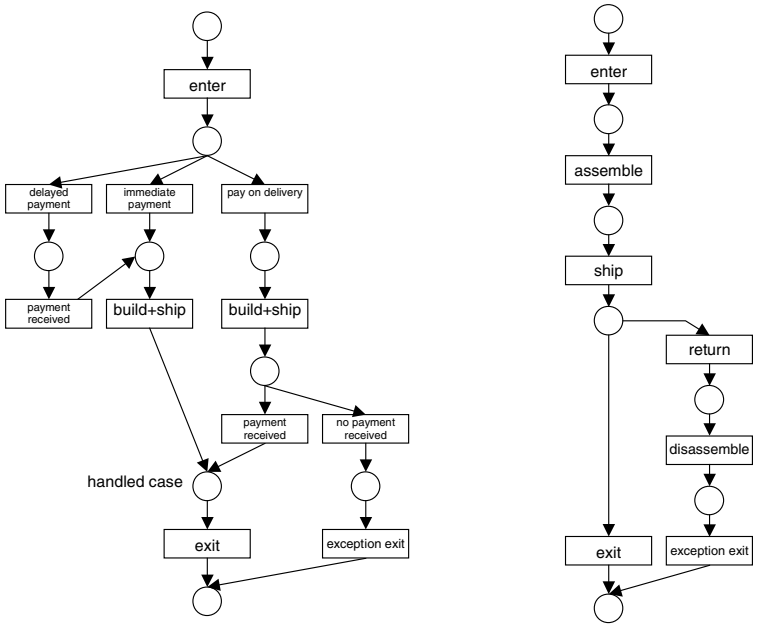

Fig. 5. Use cases "back office" and "handling"

5.2 Class Diagram

We use an adapted and extended version of the class diagram pattern for internet shops of [14]. It only models the classes involved in the customer related use cases. The elaboration of the payment and billing part of the data model is not treated.

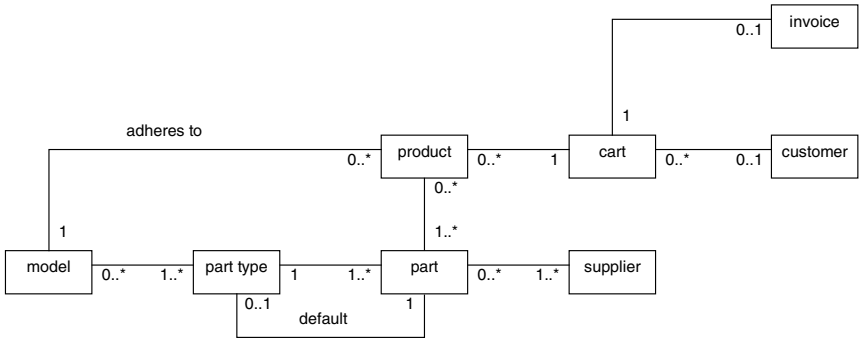

Fig. 6. Class diagram of the web shop

Each visit of a customer to the web shop involves a shopping cart. For "product" one might think of e.g. a PC configured with a number of components (parts). Each product adheres to (is configured according to) a model. Such a model has part types

(like a hard disk) that may be chosen from a number of allowed parts. Of course, a configured product may only contain those parts that are allowed by the part types of its model. In the table below, we list some attributes.

Table 1. Classes and some attributes

Class	Attribute
cart	payment mode
product	amount
	color
	price
model	assembly price
part	price
customer	name

5.3 Class Lifecycles

Each instance of a class in the class diagrams has a lifecycle. We will also use workflow nets for these lifecycles. Note that each transition in a class lifecycle is usually caused by an external event.

Many lifecycles are very simple, as shown by the examples of Fig. 7. Note that once a model has a relation with a product, one is not allowed to change it anymore. Many products may use the same model at the same time, and an update of such a model is only allowed if all products have released the model. This is modeled by having n (an arbitrary large number) tokens in the central state of the model lifecycle.

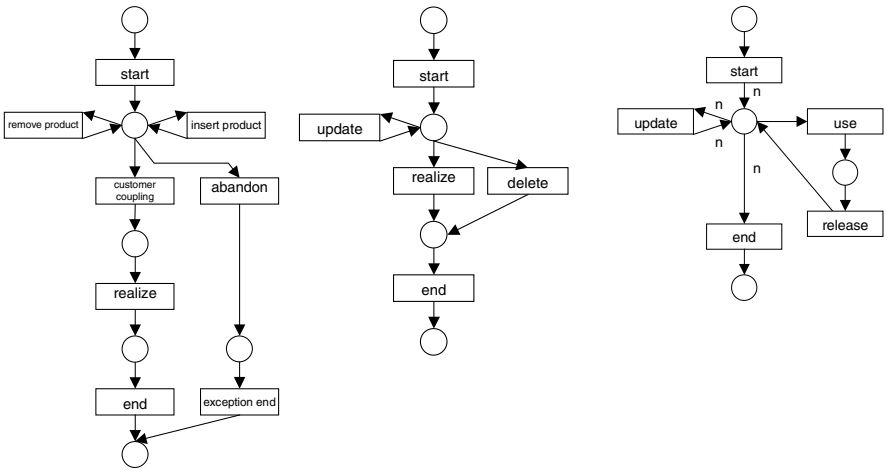

Fig. 7. Lifecycles of shipping cart, product, and model

5.4 Interactions between Workflows

The next step is to associate transitions of the use case workflows and the class lifecycles. In Fig. 8 we show how the transitions are associated.

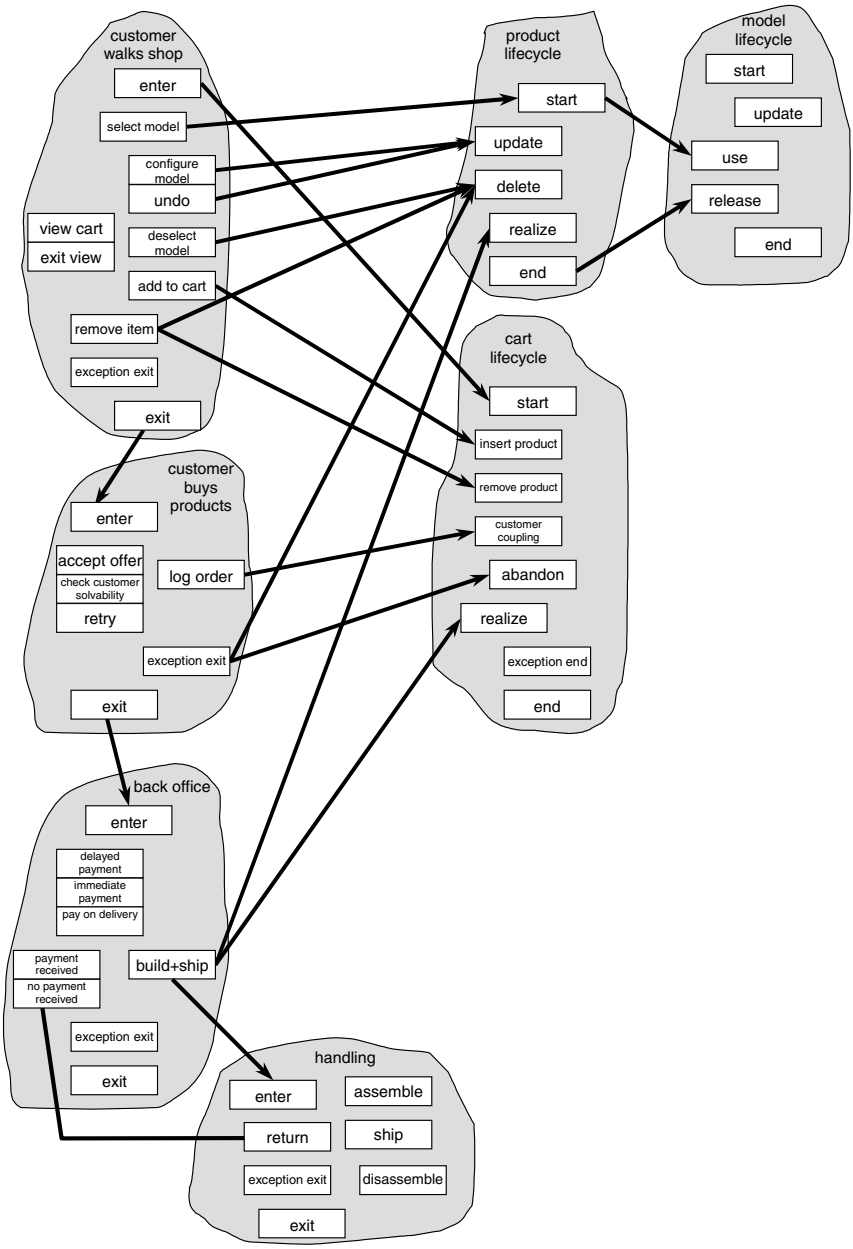

Fig. 8. Relations between workflow transitions. Use case transitions are shown on the left, life cycle transitions on the right

Relations between transitions in two use cases. A transition in a use case can start another use case. This may be the exit transition like in "customer walks shop" that starts (upon normal termination) the workflow "customer buys products". It can also be any other transition: for example, "build+ship" in "back office" starts the workflow "handling". This coupling is usually asynchronous.

A transition in a use case can also trigger a transition in another use case. For example, "return" in "handling" fuses with the "no payment received" in "back office". Here we must fuse the transitions to guarantee that "return" is only allowed to fire in case of pay on delivery.

Relations between transitions in a use case and a lifecycle. Now we look at the mapping from use case transitions to lifecycle transitions. This mapping can be synchronous or asynchronous, but will probably never be realized by transition fusion: the transitions in a data object can be triggered by many use cases.

A transition in a use case may be associated to multiple lifecycle transitions, each in a *different* lifecycle. For example, "remove item" in "customer walks shop" means that both "delete" and "remove product" should fire.

Multiple transitions in the same or different use cases may also be mapped to one lifecycle transition. For example, "deselect model" and "remove item" both mean that "delete" should fire. However, this coupling has a direction: if "delete" fires, this does not imply that also "deselect model" should fire.

Another reason for not having a one-to-one mapping might be that the modeling has been performed at different levels of abstraction: a transition corresponds to a subnet. This does not occur in our example.

Note that not all transitions of a use case have to take part in a mapping. For example, the "view cart" transition in the "customer buys items" workflow only retrieves the current contents of a shopping cart and therefore does not take part in the lifecycle of the cart.

Relations between transitions in two lifecycles. Usually, we will also have relations between transitions in different class lifecycles: if a class changes state, a change may also occur in the state of a dependent class. For example, if we create a product, the corresponding model will be locked: no one is allowed to change it anymore.

5.5 Coupling Workflow Transitions and Class Methods

In the following table, we give an informal overview of the methods that are called if a transition of a specific use case fires. In a latter stage, we have to specify exactly what the input and output parameters of each method are and how they are related to the data carried by the workflow of a use case.

In the table, we see for example that the workflow "customer walks show" has two instance variables representing global data (cart and current product). Those are filled if the "start" transitions of the lifecycles of cart and product fire, which is caused by the transitions "enter" and "select model" of the workflow.

Table 2. Some workflow transitions and related method calls of lifecycle transitions.

Workflow	Transition	Class	Transition	Method call or relation change
customer walks shop: cart, current product	enter	cart	start	(creation of new instance)
	select model	product	start	(creation of new instance)
	configure model	product	update	Change product-part relation. Update attributes of product (amount, color, and price).
	undo	product	update	Change product-part relation. Update attributes of product according to default (amount, color, and price).
	deselect model	product	delete	--
	add to cart	cart	insert product	Add relation between cart and product.
	remove item	cart	remove product	Remove relation between cart and product.
		product	delete	--
	view cart	--	--	--
	exit view	--	--	--
	excep. exit	--	--	--
	exit	--	--	--
customer buys items: cart, requested payment mode	log order	cart	customer coupling	Create new customer class if not yet existing. Add relation between cart and customer class. Update payment mode attribute.
	…..	…..	…..	…..

6 Conclusions and Future Work

We have shown how workflow theory can be applied to requirements engineering, in particular for the formalization of use cases. It is again a confirmation that the colored Petri net framework is a sound base for model integration. The approach presented here gives a systematic way to develop system specifications and the possibility to verify properties, in particular soundness. We did not have enough room here to show how this approach can be continued to decompose a system into logical components, but [12] and [5] provide a theoretical base to support this approach. Experience in several software development projects has convinced us that that approach works well.

We are working on a design method for component based development where workflow nets are first class citizens, i.e. we try to model all process aspects of a system components as workflows. We like to do this using standard techniques like use cases, sequence charts and activity diagrams as much as possible. The idea is to add some modeling restrictions (as conventions) to the existing techniques and to limit the introduction of additional notations as much as possible.

References

1. Aalst, W. van der: Verification of Workflow Nets. In: Azema, P., Balbo, G. (eds.): Application and Theory of Petri Nets 1997. Lecture Notes in Computer Science, Vol. 1248. Springer-Verlag, Berlin (1997) 407–426
2. Aalst, W. van der, Desel, J., Oberweis, A.: Business Process Management, Models, Techniques, and Empirical Studies. Lecture Notes in Computer Science, Vol. 1806. Springer-Verlag, Berlin (2000)
3. Aalst, W.M.P. van der: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, Vol. 8, no. 1 (1998) 21–66
4. Aalst, W. van der, Hee, K. van: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2002)
5. Aalst, W. van der, Hee, K. van, Toorn, R. van der: Component-based Software Architectures: a Framework Based on Inheritance of Behavior. Science of Computer Programming, Vol. 42 (2002) 129–171
6. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, Vol. 40. Cambridge University Press (1995)
7. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language. 2nd edn. Addison Wesley (2000)
8. Girault, C., Valk, R.: Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications. Springer-Verlag, Berlin (2002)
9. Hee, K. van: Information Systems Engineering: A Formal Approach. Cambridge University Press (1994)
10. Hee, K. van, Sidorova, N., Voorhoeve, M.: Soundness and Separability of workflow nets. Submitted for publication
11. Jensen, K. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 1: Basic concepts. 2nd corrected printing. Springer-Verlag, Berlin (1997)
12. Kindler, E., Martens, A., Reisig, W.: Inter-Operability of Workflow Applications: Local Criteria for Global Soundness. In: Aalst, W. van der, Desel, J., Oberweis, A.: Business Process Management, Models, Techniques, and Empirical Studies. Lecture Notes in Computer Science, Vol. 1806. Springer-Verlag, Berlin (2000) 235–253
13. Kruchten, P.: The Rational Unified Process: An Introduction. 2nd edn. Addison-Wesley (2000)
14. Fernandez, E. B., Liu, Y., Pan, R.Y.: Patterns for Internet shops. In: Procs. of PLoP 2001, http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/ebfernandez0/PLoP2001_ebfernandez0_1.pdf
15. Reisig, W.: Petri Nets, An Introduction. Springer-Verlag, Berlin (1985)
16. Sommerville, I.: Software Engineering. 6th edn. Addision-Wesley (2000)
17. Mazza, C., Fairclough, J., Melton, B., de Pablo, D., Scheffer, A., Stevens, R.: ESA Software Engineering Standards. Prentice-Hall (1994)
18. Verbeek, H., Aalst, W. van der: Woflan 2.0: A Petri-Net-Based Workflow Diagnosis Tool. In: Nielsen, M. Simpson, D.: Procs. 21st International Conference on Application and Theory of Petri Nets. Lecture Notes in Computer Science, Vol. 1825. Springer-Verlag, Berlin (2000) 475–484

A Model to Support Collaborative Work in Virtual Enterprises

Olivier Perrin[1], Franck Wynen[1], Julia Bitcheva[1,2], and Claude Godart[1]

[1] LORIA – INRIA – UMR 7503
BP 239, F-54506 Vandœuvre-lès-Nancy Cedex, France
{olivier.perrin,franck.wynen,julia.bitcheva,claude.godart}@loria.fr
[2] France Télécom R&D,
22307 Lannion, France

Abstract. Virtual enterprise gathers partners distributed in space, time and organizations, in order to achieve a common goal. Their business process realization needs the coordination of their distributed interactions. This paper presents the *Synchronization Point* model. It provides support for cooperative process management and coordination. It offers pertinent information about work progress while maintaining adequate privacy of information, and supports both long-time transactions and dynamic process definition. Then, its data repository and activity manager helps human interactions in cross-organizational applications.

1 Introduction

Business pressures (margin erosion, development costs, time-based competition) are placing increased emphasis on how organizations operate and interoperate with other enterprises to achieve a common business goal [6]. B2B interactions must take place simply, and organizations must work more directly with their suppliers and customers, to respond more quickly to changes. In the same time, rapid growth of web technologies is beginning to transform traditional inter-enterprise business models and allows virtual enterprise creation.

To enable organizations to adapt to this new business environment, a middleware is required to provide dynamic and flexible integration between partners in the value chain. Although new technologies will be needed to enable such integration, they will have to work seamlessly with both intra-enterprise and inter-enterprise existing business processes, while maintaining the privacy of information and the autonomy of participants [5,7]. In this paper, we propose a concept that tries to answer these features and an implementation of this middleware as a Web service. This work is partially developed in the context of a cooperation between France Telecom R&D (the *e-Process* project) and LORIA.

In section 2, we give a definition of a virtual enterprises and a list of awaited features for cross-organizational processes. We also introduce the idea of a process service. The *Synchronization Point* model is presented extensively in the next sections: its model is described in section 3, while its implementation is detailed in section 4. Section 5 gives a short comparison to related work. Section 6 concludes.

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 104–119, 2003.

2 Problem Statement

2.1 Virtual Enterprise

A virtual enterprise (VE) is an organization that allows enterprises to create a partnership for a specific project. To work on this project, the trading partners have to share competences, resources and processes (or fragments of processes) and make members work together. But these members belong to different physical organizations that can be in different places and different time zones.

Contrary to classical organizational structure based on long-standing business partnerships, VE organizational model has to be more dynamic and loosely coupled with the specific business partners. Another difference is that a VE achieves its objectives by defining and implementing processes that are distributed across several organizations [15]. Each partner implements a subset of activities of the overall process (partners work on different activities that can be composed) or the same activity can be implemented conjointly by several enterprises. This means that activities must be coordinated and synchronized. Another requirement is to specify each member contribution by a contract: the contract specifies deliverables, roles, responsible and contract terms (deadline, quality, guarantees...). To summarize, the virtual enterprise realization needs to take into account resources, organizational models, contracts between participants, and process models. Our virtual enterprise model is described in [1]. In this paper, we focus on the collaborative process model.

2.2 Cooperative Processes

A process definition consists of a network of activities and their relationships [20]. In a cooperative process, partners from different enterprises realize both atomic and composite (sub-processes) activities. In fact, a cooperative process definition is similar to traditional workflow in the sense that it describes the flow of the composed process. However, if they could be an acceptable solution for sequential activities, workflow systems are not fully adequate for the coordination of cooperative activities. As a matter of fact, the major characteristic of cooperative activities is that they are realized by parallel flows of execution, modeling the activity of each contributor. When using a traditional workflow system to model such a cooperative activity, time efficiency could be impaired: partners may have to wait for termination of all activities before being able to estimate result compatibility and before making a decision. For faster reaction to the changes, collaborative partners have to exchange intermediate results before the end of their contributions. Some flexible workflow systems can manage intermediate results exchange, but only if these exchanges are anticipated and modeled before the process execution. But, human collaboration activities are too unpredictable and cannot be totally anticipated. As the considered collaboration is not fully automated, and the human reaction is uncertain, a VE requires the ability for dynamic process definition change. Another consequence of the human participation is the relatively long time of their reactions. The use of

traditional ACID transactions is inappropriate for the long-running activities because of resource locking. Instead, such activities require *long-time transactions* : atomicity and isolation levels must be relaxed and intermediate results can be released before transactions end. A cooperative behavior requires relaxed forms of transactions that are hardly supported by traditional workflow systems.

On the other hand, groupware tools provide implicit coordination means. They support multiple partners parallel work by managing divergence through version storage, but since there is no explicit control of this divergence, the global data integration is delayed until the last phase. The problem is that the divergence can then be so high that the global integration is hard to achieve. To manage this problem, they provide group awareness means, which allow the participants auto-coordination [8].

Anyway, the solutions are not satisfactory and the problem remains "*how to coordinate cooperative activities ?*" The strong interdependency of partners parallel work requires intermediate results exchange and process-progress management. Our approach tries to combine the advantages of workflow and groupware tools. By adding the flexibility of group awareness and implicit coordination to the explicit coordination of workflow systems, we provide both a model and a tool that allow partners to coordinate themselves during the work progress. Moreover, we support both long-time transactions and dynamic process changes.

2.3 Process Services

To take part in cooperation projects, an organization must declare what it can offer to partners, thanks to the description of the offered products and services. On the other hand, external processes will need event feedback, in order to control their own work progress. However, business processes are part of the enterprise strategic core, as they represent the organization know-how and contain a lot of proprietary information. Since all the outcomes are the result of an internal business process, partners must not have direct access to this information. So, our *process service* is an abstract representation of the enterprise private business process. The *process service* model is a layered model, where the *process service layer* is a public abstract definition of the outcomes the enterprise is able to deliver, and the *business process layer* is the internal process flow in the enterprise. The model clearly separates the enterprise offer from its implementation in order to respect the enterprises privacy needs : no direct access to internal processes and no restriction for their implementation. Besides the outcomes (products and events) description, a *process service* definition (close to definition of [9]) contains the conditions surrounding the offers (inputs, guarantees. . .), as well as the providers information (access information, communications modes. . .), and includes information dedicated to the data and activities synchronization, the transactional management, and the retrieval of the services state [2]. Then, our *process service* definition is compliant to the WSDL standard [19]. Using this abstraction level, the cooperative process realization boils down to the problem of process services composition and integration. Interactions between process services have to be coordinated. This is the task of the synchonization point.

3 Synchronization Process Model

3.1 Introduction

Let us now introduce the *Synchronization Point* model. A SP is a cooperative cross-organizational activity that provides facilities for managing coordination and synchronization of cross-organizational processes. It provides as much interaction and parallelism as possible while trying to guarantee correctness. This requires a flexible definition of coordination and synchronization criteria at the beginning of each cooperation in order to allow cooperative process revision (addition, suppression or modification) as the work progresses.

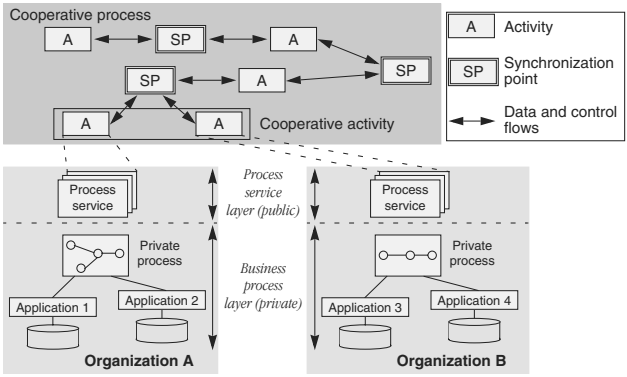

Fig. 1. Process service model and synchronization point

Figure 1 depicts the *process service* model, and introduces *Synchronization Point* (SP) processes. An organization manages its process (private process), using applications and/or workflows systems. It publishes a process service. Two (or more) process services participate in a **cooperative activity**. The idea of cooperative activity, i.e. an activity explicitly designed to support the cooperation of a group of agents, is not so developed in the context of process management. Thus, we introduced here our proper concept: a cooperative activity is handled by a synchronization point. This SP manages not only the activities as they are described in the public process service, but also the data managed by these activities. It aims to coordinate the data flow, the control flow and the transaction flow, and it manages the cross-organizational exchange and controls the contract terms: deadlines, outcomes guarantees,... Moreover, the SP also provides tools allowing partners synchronization for the realization of a common goal.

Thus, a cooperative process definition becomes a set of activities and synchronization points that coordinate partners work. To achieve this coordination, a SP implements several functions for controlling, deciding and making evolve the synchronized cooperative process. As a cooperative entity, the SP must provide object sharing between contributing organizations, communication, awareness and coordination means :

– **object sharing**. It is the minimum service to provide for cooperation support, and this also implies the management of versions and of partner privacy.

This last point is especially important in the context of cross-organizational processes. In order to preserve and ensure privacy and autonomy of participants, we must reduce objects inter-visibility (i.e. visibility of a participant object by another participant) as low as cooperation needs. This can directly influence the definition of SPs. Of course, it is not because an organization A shares an object with participant B and B share another object with participant C that A accepts to share its object with C. But also, it is not because A shares the same object with B and C that it wants to synchronize simultaneously with B and C. Clearly, inter-visibility of objects needs to be based on contracts that explicitly define privacy rules.

- **communication**. Partners need to talk to each other, to discuss, to share artifacts, when they are coordinating their processes and reconciling their process differences. This potentially includes chat, email, video-conferencing, white boards and so on. A SP must also support different communication policies (request/response, solicit response, one-way, notification...).
- **coordination**. A major difficulty introduced by distribution is the coordination of the activities of team members. We address this problem as is :
  - *task coordination* (or explicit coordination) based on the hypothesis that it is possible to define a process and to enforce this process on working sites. We address this problem thanks to a high-level specialized process inspired from the Deming virtuous circle (*Plan*, *Do*, *Check*, *Act*),
  - *group awareness* (or implicit coordination). Being aware of the partners work is a good way to help coordination of people in the realization of a common objective. To achieve this, a SP notifies partners of all events concerning shared data and activities (i.e. creation, modification, versioning, deletion). In order to provide only pertinent information and avoid overload, SP allows each participant to create a subscribed events list so that he/she will be notified only for a kind of event.

To synchronize cross-organizational processes, a SP provides:

- **check**. To be able to synchronize partners, a SP needs information on their work progress. To ensure enterprise privacy needs, it does not ask information on their internal processes, but uses only the information that are associated to process services by contract. This information contains a list of outcomes and a list of properties that outcomes must verify. Outcomes includes intermediate and final results, predefined events (e.g. end of specific task, document reception. . .). Properties includes time schedule, quality criteria, and so on. As outcomes and properties are application dependent, the model makes it possible to structure and store checks and related decisions. So, the role of a SP is to compare and evaluate the activity outcomes regarding a set of predefined criteria (presence of document, quality, schedule, objective fulfillment. . .). Depending on the result of check, partners implied in a SP can decide actions. Among these actions, we distinguish (re)planning.
- **act**. Depending on the result of *check*, a set of *actions* is decided by partners. The objective is to react and fill the gap between the process initially forecasted and the executed one. A *decision* relates an event (why), an action (what), a group of people in charge of its realization (who) and a schedule

(when). Events can be either the result of the normal execution of the process, or an exception. An action can concern one or several organizations. Decision are specified in *action rules*. For instance, what should be dowhen one partner does not respond, but most important actions undertaken during control consist in replan the process.

– **plan**. This SP function supports dynamic process changes. It allows SP revision (e.g. deadline change, action rule modification...) as well as cross-organizational process adjustment. This adjustment includes the increasing of resources dedicated to an activity, shift of the end date of an activity, shift of an activity in a process plan, switch of two activities, addition of an activity, replacement of an activity by another, suppression of an activity and so on. Of course, cross-organizational process adjustment is constrained by the need of autonomy requested by partners.

3.2 Definitions

In the following, we denote a process with upper case letter such as P, Q or R and we introduce $O(P)$ the set of objects manipulated by the process P and denote its elements with lower case letters such as x, y or z. Objects of a given process could be files, data, goals, but also sub-processes and/or activities.

General Definitions. WFMC defines a business process as a set of one ore more linked procedures or activities that collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships. In our mind, a collaborative business process represents the different steps among several organizations required to complete a given task or to achieve a given goal.

Relations. We consider that two processes cooperate if it exists a relationship between at least one of their objects. We denote this relation *rel* and use notation: $x\ rel\ y$ where x or y could be both documents or sub-processes. Without control or cooperation, two divergent versions of the same object may be obtained.

***Sat* Functions.** When a relationship exists between two objects, we associate and define a *Sat* function that is able to satisfy the relation, denoted $Sat(x\ rel\ y)$. Satisfying a relation is the result of an activity that consists in checking if x and y realizes a set of conditions upon both x and y (see 3.2, Synchronization Activities). We may use static functions to check whether the relation is satisfied or dynamic functions to ensure that it is. In most cases, we may use both to perform complex relations among process objects.

$$\forall x, y\ (x\ rel\ y) \Rightarrow \exists\ Sat(x\ rel\ y) \mid Dom(Sat(x\ rel\ y)) = \{true, false\} \quad (1)$$

As previously underlined, objects can be processes and we can write:

$$\forall P, Q\ (P\ rel\ Q) \Rightarrow \exists\ Sat(P\ rel\ Q) \mid Dom(Sat(P\ rel\ Q)) = \{true, false\} \quad (2)$$

Synchronization Point. A synchronization point (SP) is a particular process denoted $P{\infty}Q$ (pronounced "P synchronized with Q") such that:

$$\begin{array}{l} \forall x \in O(P),\ y \in O(Q)\ and\ (x\ rel\ y), \\ P{\infty}Q \Rightarrow Sat(x\ rel\ y) \in O(P{\infty}Q) \wedge Dom(Sat(x\ rel\ y)) \ = \ \{true, false\} \end{array} \tag{3}$$

A synchronization point allows for synchronizing one cooperation between two or more processes. It includes every *Sat* function that resolves relations between their objects or processes themselves. It exists a bijection among the existence of a synchronization point and the existence of relation among process objects. In other words, we assume that 1) we cannot have a synchronization point without having at least one object relation and 2) all the *Sat* functions that resolve relation among process objects of two given processes P and Q are located in a $P{\infty}Q$ synchronization point. Moreover, we will describe later how the SP is able to offer a versioned space for all these objects.

A synchronization point is a process that includes parallel activities, and each of them resolves the *Sat* function corresponding to the relations among process objects involved in the cooperation. We separate individual behaviors of processes P and Q and the behavior of their common synchronization point. In the same way, if a single process P owns the objects involved in a relation, we can consider the corresponding *Sat* function as a sub-process of $P{\infty}P$:

$$\forall x, y \in O(P)\ (x\ rel\ y) \Rightarrow P{\infty}P\ exists \quad (\text{with } P{\infty}P \neq P). \tag{4}$$

Synchronization Activities. A synchronization activity is an activity that tries to resolve a *Sat* function. It is included in a synchronization point and is denoted $\frac{P{\infty}Q}{Sat(x\ rel\ y)}$ where $Sat(x\ rel\ y)$ is the *Sat* function that resolves the relation of two objects $x \in O(P)$ and $y \in O(Q)$. A synchronization point includes the *Sat* functions and the objects involved in relations between processes. Now, we can define the object set of a SP $P{\infty}Q$ as the set of all shared objects of P and Q for which it exists both a relation and a *Sat* function:

$$\bigcup_{i=1}^{n} x_i \in O(P) \cup \bigcup_{j=1}^{m} y_j \in O(Q) \cup \bigcup_{k=1}^{n*m} Sat_k(\bigcup_{i=1}^{n} x_i\ rel_k\ \bigcup_{j=1}^{m} y_j) \mid (x_i\ rel_k\ y_j) \tag{5}$$

When a synchronization activity failed (the relationship between the two objects or the two processes can not be satisfied by the *Sat* function and. the *check* has failed), the first option is to accept that the *Sat* function could be unsatisfied. A human decision is needed and partners of the collaborative process are notified that a divergence could exist between the objects. The second option is to start the replan activity. This activity is dedicated to process execution and activities in $P{\infty}Q$ must either be retriable or compensatable.

Visibility. In order to preserve and ensure privacy and autonomy of process participants, we must reduce process object inter-visibility as tiny as cooperation needs. In a relationship between two processes P and Q such that P is a sub-process of a process R, $P \in O(R)$, and Q is a sub-process of a process S,

$Q \in O(S)$, we may have $(P\ rel\ Q) \Rightarrow P{\infty}Q$ but not $R{\infty}S$ in order to preserve autonomy of processes R and S. We consider a workflow management system activity as black box due to the fact that some workflow managements systems do not respect the WAPI [21] specifications. Thus, to build a relation among a workflow activity wa and a process P, we must before encapsulate wa as an object in a process Q and then declare the cooperation such as :

$$\forall P, wa\ (wa\ rel\ P) \Rightarrow \exists Q \mid wa \in O(Q) \wedge Sat(wa\ rel\ P) \in O(P{\infty}Q) \tag{6}$$

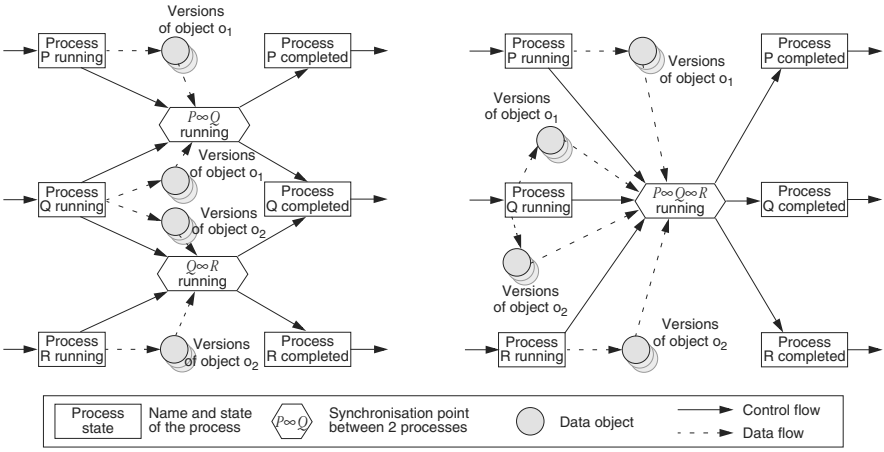

Fig. 2. Two synchronization points

Multi-process Relationship. We have defined above the meaning of $x\ rel\ y$ and its consequences among processes that include x and y objects. We now consider relations among three processes to evaluate the meaning of $P{\infty}Q{\infty}R$. First, we must keep in mind that processes could be part of different organizations. At process level, we have as many synchronization points as process relationships. Figure 2a shows a situation where process Q is involved in two cooperative processes. Thus, P and R are not aware of each other, preserving their autonomy. In fact, we define a privacy rule where a process is involved in a synchronization point if and only if a relation exists among one of its objects and at least one object of the synchronization point. We write :

$$\forall x \in O(P_1),\ x \in O(P_1{\infty}P_2{\infty}...{\infty}P_n) \Rightarrow (\exists y \in O(P_2{\infty}...{\infty}P_n) \mid x\ rel\ y) \tag{7}$$

Definition 7 exposes a recursive way to involve a process in a cooperation. This allows detecting when visibility agreements are needed for an incoming process. Thus, we are allowed to propose a synchronization point model where we share "*all necessary but no more and at least, under the control of each participant*". In reference of the previous example, if a relation exists among o_1 and o_2 and if P and R accept to share those objects with each other, we can merge our two synchronization points in a single one. Figure 2b depicts a situation where P, Q and R work together.

3.3 State Model

In paragraph about visibility, we discussed the differences between process and object relationships. In order to coordinate both objects of two processes and activities belonging to these processes, we introduce the synchronization point for managing the exisiting relations between objects or processes, and providing a coordination based on process state and coordination based on object state.

Process. Our process state model is WfMC compliant [20]. We use the defined states: *Initiated* (I), *Running* (R), *Suspended* (S), *Terminated* (T), *Active* (A) and *Complete* (C). Thus, we denote the current state of a process or an activity with an exponent letter (e.g. P^T means that the process P is terminated). A compensatable process implies first the abortion of the execution of the process instance. A process is retriable when the instance can be invoked a finite number of times such that the last invocation commits while all the previous ones abort.

Object. For object, we introduce a new state model.
States
Defined (d). The object is defined but empty. We can define relations among objects and therefore may create synchronizations point. This is generally the state of an object that is involved by a process at design time.
Instantiated (i). The object now exists in a process repository but is not yet manipulated. It must be the state of a newly created object or the representation of a binding between an object and its envelope.
Modified (m). The object is currently modified by one ore more processes or activities. The process that manipulates the object may read in and write to its own repository.
Read (r). The object is in read mode. That is, if an object is involved in a relation, we may have a synchronization activity even if it does nothing else than checking the object availability. This is a final state.
Delivered (v). The object is posted to the SP repository whose process depends on. From the process point of view, the work is done and checks can be started.

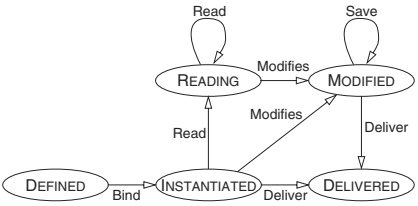

Fig. 3. Object state model

Transitions
Figure 3 depicts the object state model. However, we assume that some transitions must be unavailable for some object types (e.g. a process cannot modify an automatic activity such as workflow but just can read its state). We distinguish 1) the *save* transition that consists in storing the object in its repository during process time and 2) the *deliver* transition that achieve a modification

session inside a process and proposes a result (which is know as intermediate result as the two processes involved in the collaboration do not yet commit) to the community involved in a synchronization point.

3.4 Semantics and Behavior

We have defined a process model, introduced the synchronization point concept and described process and object state model. In this context, we define a language in order to describe relations between objects and processes.

We propose the notation *(object or process)*^{State} $\Rightarrow$ *Transition (object or process)* to express that a transition will depend on an object or process state[1]. We assume that the left part (i.e. a pre-condition) is a boolean expression and must be the result of *((object or process)*^{State}*==State) = true*. Thus, we extend pre-condition as well to conditions upon dates such that *(date)*$\Rightarrow$*Transition (object or process)* where *(date)* means *((predefined date)*$\geq$*(now))= true*. A clearest notation is :

$$P^{State} \Rightarrow \textit{Transition (Q)} \text{ (i.e. if P is in state "State" then transition on Q)} \quad (8)$$

Moreover, the right part is an action that involves an object or a process in a transition as defined in a state transition model. For example, if we ask for a process Q to run when another process P will complete (i.e. such as a sequential routing), we denote $P^C \Rightarrow Run(Q)$. That corresponds to these properties :

1. C is a state for P object type
2. State of P is equals to $Complete$
3. Run is a valid transition for Q object type
4. State of Q is compatible to a Run transition (i.e. Q is *initiated*, *suspended*, *active* or *running* and Q is a process)

If all the conditions above are satisfied, Q is now in *running* state. We are able to distinguish explicit conditions that compose a scenario and implicit ones that return exceptions. Thus, properties 1 to 3 check for inconsistency at process definition step while the later allows for capturing process errors at runtime.

3.5 Cooperation Contracts

A necessary condition to the existence of a synchronization point is the definition of at least one object relation. But with three or more objects, relations may be complex and should generate more than really necessary synchronization activities. When an object is involved in several relations at the same time, we must synchronize it in a single synchronization point or we must define two or more synchronization points in order to serialize the modifications on the object. We assume that :

$$\begin{array}{l} \forall x \in O(P), y \in O(Q), z \in O(R), \\ x \; rel \; y \wedge x \; rel \; z \Rightarrow P \infty Q \infty R \vee (P \infty Q) \infty R \vee (P \infty R) \infty Q \end{array} \quad (9)$$

[1] It is allowed to use both "." (logical and) and "+" (logical or) to express WfMC operators.

For example, in order to perform a three-part cooperation, we take interest in Q and R object inter-visibility. Relations $x\ rel\ y$ and $x\ rel\ z$ mean that P and Q for a part and P and R in another have accepted to share information but not this is not true for Q and R yet. In fact, we assume that a contract exists between P and Q and another exists between P and R. That is, if those contracts include privacy protection clauses (e.g. protection of a part of a document), we must not provide entire visibility to R in Q and vice versa. Thus, faced to a multi-relational description, we must test whether :

- contracts exist and allow users to share their objects in a common synchronization point. This case means that a contract also exists between Q and R and allows involving P, Q and R in a single synchronization point.
- contracts do not exist and we create synchronizations points for each independent cooperation.
- contracts do not exist and process participants meet for a deal. In fact, we propose new contracts in order to reduce the number of SP.

Therefore, if $x\ rel\ y$, $x\ rel\ z$ and Q must process during R we are allowed to propose a global synchronization point if and only if both Q and R participants accept to work together. Otherwise, we must serialize two different synchronization points, selecting for example $P \infty Q$ and pushing its result in a cooperation with R such as $(P \infty Q) \infty R$ (i.e. different from $(P \infty Q \infty R)$).

Thus, we enlighten here the strong link that exists between relations among objects and contracts among organizations that own them and we claim that a relation among objects corresponds to a contract and a *Sat* function. The first represents work group awareness through process participants, privacy policy associated to objects and communication means description and the second proposes tools designed to resolve relations among objects.

3.6 Repository and Versioning

For each process (i.e. P), we define a repository where objects involved in the process (i.e. $O(P)$) are stored during processing time. Process activities use this repository to manage objects. For each synchronization point, it exists a repository that allows all the process participants involved in the cooperation for sharing objects. So, given two processes P and Q that collaborate we may have three repositories : $O(P)$, $O(Q)$ and $O(P \infty Q)$. All the repositories encapsulate objects and give the ability to define privacy policy in a multi-organizational virtual enterprise. Last, a repository hides objects inside a process scope.

Different versioning policies exist for synchronization point and for processes. In fact, a SP repository is considered as a contractual view of one or more organizations version spaces while processes and activities repositories are considered as workspaces dedicated to evolution of data. That is, synchronization points, individual processes (those which are not involved in a cooperation) or merely processes that manipulate some relation independent objects (objects that are not shared) have a direct read and/or write access to organizational repositories. Therefore, they use the versioning policy of the organizational repository they check in or they check out. On the other hand, processes manage independently

their own versions in their own repository. So, version mismatches induced by this policy are hidden by privacy rules on process repositories. Moreover, this avoids an exponential amount of version synchronization messages between organizations over Internet. When a synchronization point completes, version spaces of the processes involved are reconciled, and the synchronization point version space is reconciled with the organizational repository. This schema for version management also applies for nested processes and their respective repositories.

4 Implementation

In our architecture, a SP starts when each partner decides to start a cooperative activity with others in order to fulfill a given objective. Each partner only describes its activities, and contracts are established to express conditions and terms of data exchange and share. Then, each partner works in autonomy, having the ability to deliver ongoing version of its work to others and coordinating its activities with the synchronization point.

The current implementation is based on a distributed architecture that uses the web services technology. Each partner hosts a part of the SP repository and exchanges are done thanks to SOAP messages. Thanks to the level of abstraction introduced for describing organization processes, the architecture does not require any specific way of doing things, such as describing precisely the organizational model or the process model. Moreover, it is not mandatory to have ÒworkflowÓ enabledÓ applications and we do not require an explicit enterprise model. In fact, we use late binding to couple one activity to respectively an application, a participant, a workflow engine or a back-end process. The only requirement is the description of *process services* and we use WSDL [19] for describing properties of processes.

A SP is managed by a tier (which can be viewed as a *broker*) that allows for storing organizations end-points, projects, abstract descriptions of the processes, contracts, roles and all the information about synchronization points. A contract is a XML document that helps us to filter what is the right information to provide to the right participant at the right time by setting up the exchange between two or more partners. A contract references a set of filters, which are also XML documents used to describe information that can be shared with others. Once being processed, the reference of the original document is delivered to the SP repository thanks to a SOAP message, describing the organization endpoint which delivers it. When another organization wants to access shared data, it requests the tier which is in charge of maintaining which partner (organization end-point) in the cooperation is able to deliver this data. In fact, the SP repository is distributed over all the participants of the cooperative activity. Once the tier finds the owner, it gives to the provider the end-point of the requester and the associated filter. Then, a SOAP based peer-to-peer communication is engaged between the requester and the provider. Of course, this communication is compliant to the existing contract between the two organizations.

Figure 4 gives an overview of the architecture. We can see on the figure that the private part of the process can be coupled with any internal application. Then, the process involved in the SP is published as a process service (aka a

web process service). In the same time, a public repository is dedicated to the data stored within the SP repository for this particular organization ($P{\infty}R$ in the figure). When an object is delivered to the SP, it is in fact stored in this particular repository and the reference of the object is sent and stored in the tier repository. The service provider is a web service that gives the ability to communicate with others thanks to messages that are compliant to the W3C Message Exchange Patterns (MEP) we have defined : the MEP *deliver*, the MEP *download* (which uses the JNLP technology), and the MEP *read*.

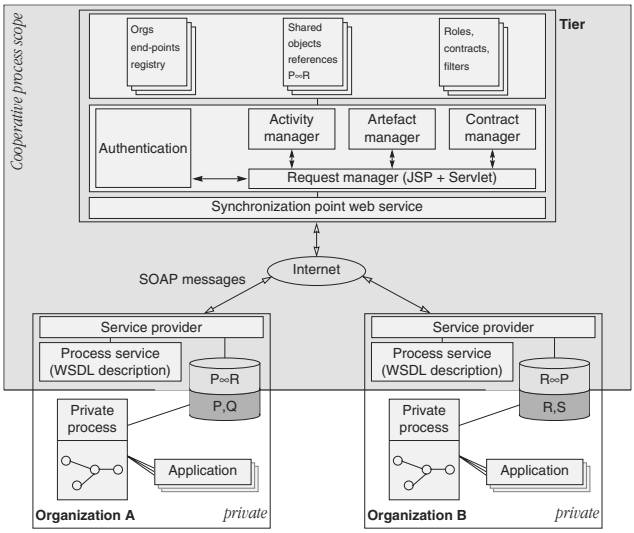

Fig. 4. Architecture diagram

Control flow is synchronized thanks to synchronization activities (*check*, *act*, and *plan*) belonging to the SP. This means that when an activity is achieved (the service provider notifies the tier that a given *process service* is completed), its results are delivered to the SP repository and when coordination is done, next activity starts if and only if all the checks (as defined in the contracts) are valid and the reconciling operations on shared objects are done. Data flow is managed thanks to a configuration and version management system named Toxic [16]. During execution of a multi-partner cooperative process, we manage all the different versions of artifacts.

By providing a shared space for versions associated with the SP (where versions of artifacts are delivered), we are able to keep one participant of this SP in touch with work progress of others. By having access to intermediate results, each participant can see what others did on shared objects. As objects are under version control, one can see previous revisions, see differences between versions and who changed what and when. Of course, this asynchronous awareness can be considered as simple, but we believe that it can help to ensure that everyone has up to date information about how the project and the collaborative activities make progress. Moreover, each participant keeps all the control on his objects, a mandatory feature for all the partners.

Finally, another feature provided by the new prototype is the ability to trace the work that is done during both autonomy periods and collaborative periods. By storing main states of all the activities, duration of these activities, missed deadlines or decisions that are taken, we provide a view of what happened and when, and where the process failed or succeeded.

5 Related Work

Current work in the research field on automating processes is not directly applicable in the virtual enterprise domain. In fact, there is no common shared middleware that fits in the needs for spreading across organizations boundaries. Current propositions lead to tightly couple one organization with another both at the architecture and process levels. Moreover, transactions models and coordination needs are not accurate. For instance, locking access to shared resources is not desirable as one organization risks to loss its autonomy. Then, recovery operations could not be under the responsibility of only one organization. To summarize, two contradictory needs coexist : the autonomy of partners and the need to get some information about processes hold by the others.

Our proposition is quite different from existing workflow systems such as CrossFlow [11] for instance, where organizational model must be replicated and where applications must be Òworkflow enabledÓ in order to be able to extract any necessary data the system may need.

Van Der Aalst [17] has proposed a model that is compliant to the WfMC model. It uses high-level coloured petri nets to model workflows. It is a global model that can be split into different parts and that provides an expressive notation that allows to model aggregation, loops, branches, concurrency, and time constraints. The approach also builds on a lifecycle model, and algorithms are introduced to determine the soundness of a given net. However, the approach is top-to-bottom and it does not allow to model exceptions, nor compensation. It also lacks the notion of roles and the model does not take into account communication between the partners.

In Weske [22], there are some ideas to resolve flexibility in workflows management systems but multi-enterprises processes are not in the scope of this reflection. One interesting idea is the definition of a meta-schema for workflows and the use of graphs for defining workflows.

The work of Casati [4] describes data exchange and process interoperability, but in a B2B context, where exchanges are limited to peer-to-peer conversations. The concept of traces is very interesting and we should include this kind of mechanism in a future version of the prototype.

Georgakopoulos [9] presents in CMI the concept of window of opportunity which allows for conciliating prescribed activities inherited from WfMS and optional activities inherited from groupware applications. Our synchronization point concept helps to provide the same kind of feature, but it wants to be more general and flexible, and gives the opportunity to exchange intermediate results.

Some works around workflows are available. XPDL [20] is used to define processes but compensation cannot be modelled. BPML [3] or BPEL4WS [12]

can be used to query the states and control the execution of process instances. PIP supports interchange between two or more organizations following Rosetta Net standard [14]. These propositions fail to provide solutions for either long term transactions, collaboration phases, or binding to internal processes.

6 Conclusion

In this paper, we propose a model and an architecture that support collaboration and cooperation for multi-enterprises processes. This work tries to offer advantages of both workflow management systems and groupware systems. With the SP model, different partners can work together and they only need to define cooperation rules corresponding to contract specification. Then, the SP is able to coordinate the work progress, and it provides all the participants with accurate information on work evolution. By including data and control flow management functions in SP, we allow a flexible process definition. We are able to adjust the collaboration process definition by updating the SP during work progress and it is possible to evaluate the cooperative process against a set of given criteria.

The first feature of the SP model is the platform and system independence, thanks to a certain level of abstraction. Thus, it is easy to integrate the model with existing back-end systems of organizations and it is not tightly coupled with workflow-enabled systems. Another feature of the model is the adherence to component-based design and composability. With this model, it is possible to compose several processes viewed as components in order to obtain an overall composed process. This leads to the ability to easily manage multi-enterprises processes. We are currently evaluating the model against the workflow patterns proposed in [18]. Then, when you need to cooperate in order to achieve a given objective, you need to exchange and to provide to others intermediate results, breaking the ACID properties of classical transactional models that are used by these WfMS. The SP model tries to break these limits.

Flexibility, portability, and interoperability are also supported and the SP model offers an effortless enterprise information systems integration, while preserving autonomy of each partner. We do not impose to be compliant to any model (both process or organizational models) in order to use our model and our architecture. This is hardly possible when we try to do this using WfMS. Then, we adopt a service-oriented architecture, and we rely on standards such as SOAP, WSDL and XML. This allows for accommodating and fitting both the architecture and the model without to throw away all the work ever done and to benefit from future enhancements.

References

1. J. Bitcheva and O. Perrin. Virtual Enterprise Meta Model. *Technical report, Vision e-company Research Program*, France Telecom R&D, April 2002.
2. J. Bitcheva, O. Perrin and C. Godart. Cross-Organizational Processes Coordination. *Technical Report LORIA A02-R-046*, Nancy, May 2002.
3. BPMI. *Business Process Management Initiative.* www.bpmi.org.

4. F. Casati, M. Sayal, U. Dayal and M.C. Shan. Integrating Workflow Management Systems with B2B Interaction Standards. In *ICDE 2002*, February 2002.
5. Q. Chen, U. Dayal, M. Hsu and M. Griss. Dynamic Agents, Workflow and XML for E-Commerce Automation. In *First International Conference on E-Commerce and Web-Technology (EC'2000)*, UK, 2000.
6. A. Dan and F. Parr. Long running application models and cooperating monitors. In *HPTS workshop*, Asilomar, CA, 1999.
7. U. Dayal, M. Hsu and R. Ladin. Business Process Coordination: State of the Art, Trends, and Open Issues. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases*, September 11–14, 2001, Roma, Italy.
8. P. Dourish and V. Bellotti. Awareness and Coordination in Shared Workspaces. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, ACM Press: Toronto, Ontario,1992, 107–114.
9. D. Georgakopoulos. Collaboration Management Infrastructure (CMI): advancements in Coordination, Awareness, and Integration. November 2001.
10. C. Godart, O. Perrin and H. Skaf-Molli. Cooperative Workflows to Coordinate Cooperative Asynchronous Applications in a simple Way. In *International Conference on Parallel and Distributed Systems (ICPADS '2000)*, July 2000.
11. P. Grefen, K. Aberer, Y. HoffnerC and H. Ludwig. CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises In *International Journal of Computer Systems Science & Engineering*, Vol. 15, No. 5, 2000.
12. F. Leyman. *Business Process Execution Language for Web Services*, 31 July 2002.
13. O. Perrin and C. Godart. A Mail Based and XML based Protocol To Support Workflow Interoperability. In *AI 2002*, February 2002.
14. Rosetta Net. *PIP, Partner Interchange Process, Rosetta Net Implementation Framework*. www.rosettanet.org.
15. H. Schuster, D. Georgakopoulos, A. Cichocki and D. Baker. Modeling and Composing Service-Based and Reference Process-Based Multi-enterprise Processes. In *CAiSE 2000*, LNCS 1789, 2000.
16. Toxic. *ECOO group*.
17. W. Van der Aalst and K. Van Hee. Workflow Management: Models, Methods, and Systems. MIT press, Cambridge, MA, 2002.
18. W. Van der Aalst, A. ter Hofstede, B. Kiepuszewski and A. Barros. Workflow patterns. www.tm.tue.nl/it/research/patterns, July 2002.
19. Web Services Description Language (WSDL) 1.1. *W3C*, March 2001.
20. WfMC. *XML Process Definition Language – XPDL 1.0*, December 2002.
21. WfMC. *Interface 2: WAPI Specification*, WFMC-TC-1009, Version 2.0, July 1998.
22. M. Weske. Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In *Proceedings of the 34th Hawaii International Conference on System Sciences*, 2001.

Towards a Library for Process Programming

Guangxin Yang

Bell-Labs Research, Lucent Technologies
600 Mountain Avenue, Murray Hill, NJ 07974 USA
gxyang@research.bell-labs.com

Abstract. Process programming is regarded as a critical approach in many cooperative process related areas including *software engineering*, *workflow management*, *business process management, etc.* Many process models, languages, and corresponding runtime support systems have been developed. We argue that a comprehensive library for process programming is essential for the acceptance, popularity, and success of this new programming paradigm. We define an architecture of such a library and present some mechanisms on how the architecture is implemented in the context of P, a process language and system for developing integrated cooperation applications.

1 Introduction

Process programming is a vital approach in a number of process management related areas, such as workflow management, process-based software engineering, business process redesign and reengineering. It concerns primarily with modeling and describing formally various aspects, e.g. *activities*, *artifacts*, *roles*, *tools*, and *the interrelationship among them*, of an often complex and long duration procedure within which a group of people cooperate with each other to accomplish a certain task [*Curtis et al. 1992*; *Salimifard et al. 2001*]. The description, or a process program, is critical to understand, design, simulate, optimize, and support processes in real world [*Fuggetta 2000*].

There have been numerous process-modeling techniques, among which language-based ones are dominant. For example, in the workflow area, people have been modeling workflow process with various *Petri net*, *directed graph*, *formal grammar*, etc. based formalisms; in process-centered software engineering, more than a dozen process languages [*Ambrola et al. 1997*], including *APPL/A* [*Sutton 1996*], *SPADE-1* [*Bandinelli 1996*], *HFSP* [*Katayama 1989*], *CSPL* [*Chen 1997*], MELMAC [*Gruhn et al. 1992*], Merlin [*Peuschel et al. 1992*], E^3 [*Jaccheri 1998*], EPOS [*Conradi et al 1994*] have been developed; in business process area, representative work includes *Process Handbook* [*Malon et al. 1999*], the GED framework [*Katzenstein et al. 2000*], and *Process Access Library* [*Hart et al. 1992*] for systematic business process modeling and analyzing. These efforts have produced more and more mature understandings of processes and mechanisms to support them.

Process programming is perhaps based on the understanding that "*business processes are software too*", a generalization of the well received proposition "*software processes are software too*" in the software process community [*Osterweil*

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 120–135, 2003.
Springer-Verlag Berlin Heidelberg 2003

1987]. Though the power of software reuse is self-evident and many have recognized that it is critical to share and reuse process programs among different systems to promote efficiency and interoperability [see some position papers in *Proc. of IEEE International Software Process Workshop, Dijon, France, 1996*], few systematic efforts have been put into developing advanced mechanisms that enable such sharing and reuse, perhaps partly due to the wide variance in the formalisms, capabilities, and semantics of process languages. We believe that research on mechanisms for process reuse will not only be a key enabler to the popularity of process technologies, but also will lead to better understandings of and more flexible support for processes.

The purpose of this paper is to propose an architecture of reusable process libraries in general and to present our implementation of in the context of the P language and system, which is designed for developing cooperative applications [*Yang et al. 2001*], in particular. Our findings include:

- Process inheritance, which bases one process definition on another, should be the primary pattern for process reuse. With explicit process inheritance, process programmers would be able to exploit the power of abstraction-specialization in developing process programs.
- Process libraries are conceptually a superset of software libraries. Though the main purpose of process libraries is to provide reusable process definitions, software components used at different stages during instance enactments should be also included in process libraries and should be able to be easily expanded.
- Process enactment services, such as creating and manipulating process instances, retrieving the runtime information about instances, should be available via process libraries. The reflection ability to access these services from within process programs would greatly increase the flexibility of process modeling.

The rest of this paper is organized as follows. In the coming section, we will propose a process library architecture. The third section details how the architecture is implemented in the context of P. Comparisons with related work are presented in the forth section. Conclusions and plans for future work are outlined in the last section.

2 An Architecture of Process Libraries

Process libraries should contain the components that can be used as building blocks for writing process programs. Obviously, people want to find reusable process definitions from process libraries to build their own process programs. The problem is once they have these definitions, how they can reuse them efficiently. Inheritance has been a very powerful way for software reuse [*Krueger 1992*]. Since cooperation processes can be programmed, introducing inheritance to process programming should boost dramatically the productivity and popularity of this programming paradigm.

At the same time, cooperative processes are a combination of both coordination and computation. Although there are strong trends and evidences that in the design of some cooperation languages people tend to make a separation between these two parts [*Cortes et al. 1996*], from the point of view of constructing process libraries, we think both parts are of equal importance. This is because coordination, in some sense, relies

on computation. For example, people need to use some kind of software tool, e.g. *a word processor*, *a compiler*, or access some resources, e.g. *a database*, *a file*, to get their work done. Therefore, elements for accessing these services should be an integral part of process libraries.

On the other hand, process programs differ from ordinary software programs in that the former is not directly scheduled by operating systems, but by process systems, which provide functions for managing process instances. Process management services of operating systems, such as creating or terminating processes/threads, are available to any program via certain system calls. Unfortunately, instance management services of process systems are not generally available from within process programs. This prevents process programmers from using these services to build more flexible process programs. Therefore, elements that provide accesses to these kinds of computation services should also be included in process libraries.

An architecture of process libraries based on these considerations is depicted in Fig. 1. We identify three critical building blocks, i.e. reusable process definitions, components for accessing computing services both external and internal to process systems. Each part should be easily extensible, though not necessarily in the same way. Since a complex process may involve any computation, a library for that computation may be included as part of process libraries. It is in this sense that we say process libraries are a superset of classical software libraries. The next three subsections will discuss some of the most important issues within each of the three parts. Our solutions to these issues are presented in Section 3.

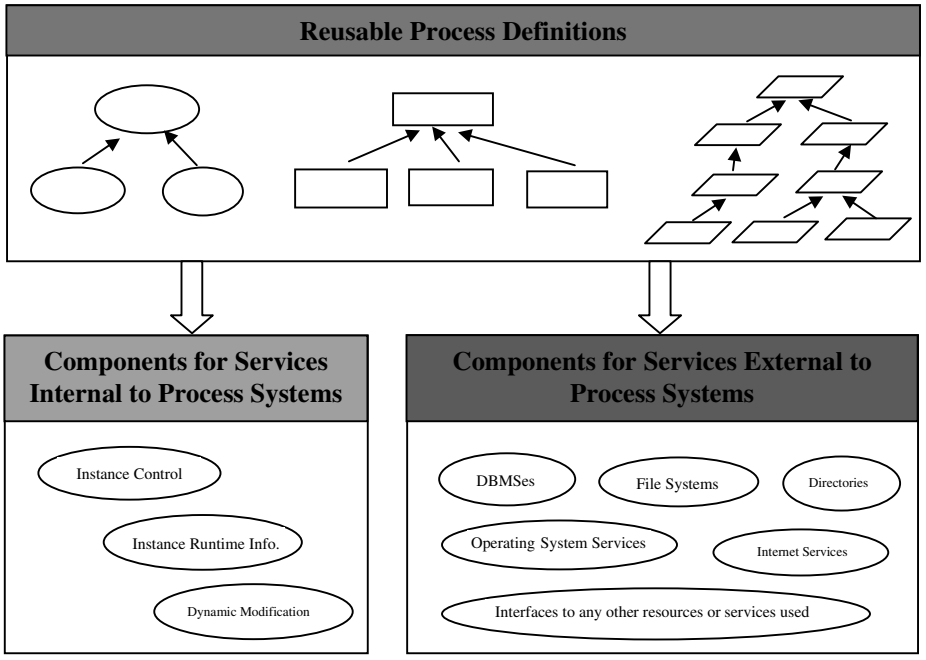

Fig. 1. An architecture of process libraries.

2.1 Reusable Process Definitions

Process inheritance has completely difference semantics, as compared with class inheritance in today's dominate object-oriented programming (OOP). Class inheritance focuses on specializing *object states* and *ways for changing these states*, which are described with member variables and methods, while process inheritance should focuses on specializing *task decomposition and descriptions* and *control and data flows among different tasks*. In the following discussions, if a process inherits another one, we call the former a *subprocess* and the latter its *superprocess*.

Task decomposition can be specialized by adding new task definitions to subprocesses. Task descriptions can be specialized by adding more details and/or overriding it with a new task definition. Control and data flows can be customized by changing the contents exchanged and conditions under which the flows occur. Of course, the specifics depend heavily on the underlying process models of process languages. From the perspective of superprocesses, some mechanisms for hiding these kinds of information from their subprocesses are desirable.

To support the new semantics of inheritance, process systems need to develop new mechanisms both for process description and process enactment. We can borrow some ideas from OOP. For example, we can define a task in a process definition as *private, protected,* or *public* to define explicitly its accessibility to its subprocesses. However these ideas cannot be copied intactly, as the mechanisms for enacting process instances are quite different from those for executing compiled programs. In Fig. 1, process definitions are organized into *trees* according to their inheritance relationship. Here only single inheritance is shown, though it is also reasonable to support multiple inheritance.

2.2 Components for Accessing Services External to Process Systems

A process system cannot stand on its own. Instead, it must talk to the outside world to get information as inputs for instance enactment and to write data to other systems for sharing or recording purposes. Consequently, we must develop appropriate mechanisms so that process programs can access various external resources, e.g. *operating systems*, *file systems*, *databases*, *directories*, *Internet services*, or *even other process systems*. Components for accessing these resources in process libraries are virtually unlimited due to the wide application areas of process technologies.

In designing these mechanisms, we need to take into account a number of factors to address the diversities of external services. We discuss two of them here. The first one is the representation, i.e. the way in which external services are represented in a process system. For example, HFSP has a *tool interface* construct; CSPL has a very similar *tool* unit construct. However, from a more general point of view, we believe that the mechanism should be extended beyond handling only standalone programs to dealing with external computing services in a truly programmable manner by utilizing the application programming interfaces exported by these services, if any.

The second factor is the protocol via which a process system talks to the outside world. In whatever the case, it is the responsibility of the process runtime system to interpret a process program and to communicate with external computing services when needed, during which a specific protocol may be followed. Since the protocols used by external services may vary greatly from low level protocols like *TCP/IP* to

high level standard ones such as *RPC*, *DCOM*, and *CORBA* or proprietary ones such as *Sun Tooltalk* and *DEC FUSE*, the mechanisms should be able to accommodate to these diversity.

2.3 Components for Accessing Services Internal to Process Systems

Process systems implement various process enactment services and hold dynamic information about process instances. These services and information could bridge the somewhat artificial gap that usually exists between the runtime and build-time functions of process systems. We argue that they should be made available to process programs so that process modeling could be more flexible. We believe that by introducing these services and information into process programming, the rigidity of process programs can be significantly decreased and they can be more flexible and adaptive to the ever-changing work conditions.

Firstly, process modeling can be more flexible by using in process programs the runtime information about process instances, which includes *when an instance is created*, *who created it*, *which state it is currently in*, *who are the participants of an activity*, etc.. For example, we can restrict the prospective participants of an activity to be the actual participants of another activity; we can force a program-defined action after a certain time period by using the creating time of an activity or process instance.

Secondly, accessing to instance enactment services gives process programs more control over process instances. Process systems usually treat process instances as its first-class citizens. Though they usually implement all the mechanisms for creating and controlling process instances, few of them support accessing these services from within process programs. Lacking this ability does no harm for toy applications. However, complex real world processes do need this kind of control to manage the creation and enactment of other process instances [*Katayama et al. 1991*].

Thirdly, for process systems that allow dynamic modifications to process definitions, accessing to this functionality from within process programs would make these programs more adaptive to various exception and increase their flexibility. Adaptivity and flexibility have been a hot topic in workflow research [*Klein et al 1998*] and recognized as future direction for software process research [*Fuggetta 2000*]. Existing research has not addressed the possibility and advantage of modifying process programs from within themselves. Our experience has shown that this capability is very useful.

3 Implementations in the P Language and System

We have outlined the architecture of reusable process libraries and discussed some of the important issues in designing these libraries. This section presents how the ideas are implemented in P. We first give a quick introduction to it as the background.

3.1 P: A Brief Overview

The basic construct of a P process is an *activity*, which holds the latest result produced towards the accomplishment of a certain task, e.g. *writing a document or a piece of code*, *filling a form*, or *drawing a picture*, and which contributes, in certain aspect, to the accomplishment of the process. The result can be accessed and modified by cooperators, either exclusively or working together with the applications developed based on the P runtime system.

The P language provides the constructs for modeling various aspects of a process, e.g. its process structure, the rules regarding message exchanges among activities, and the structure and operations of the results held by its activities. More specifically, P has the following four key constructs:

- A *class* provides a full-featured object-oriented language for defining the structures and operations of the results produced in activities and data and control information passed among them.
- An *activity* describes a task that is expected to produce a specific result, which is modeled as a class and is called an activity object (denoted as Õ). An activity can also be a *process*, which we call it a *nested process*.
- A *trigger* is an event-condition-action rule for exchanging both data and control information among activities. Conditions and actions are expressions based on activity objects and other global and predefined objects.
- A *process* describes how a higher-level task should be enacted by grouping a number of activities as a logical unit.

The following code describes a review process. For simplicity, we omit the details of classes *CDocument* and *CReview*. The process defines two activities, i.e. *AProposal* and *AReview*, which produce a *CDocument* and a *CReview* object respectively. The trigger in *AProposal* states that after this activity is finished, the generated document will be visible to activity *AReview*, which will then be started so that a review could be generated. The trigger in *AReview* states that after a review is generated, the summary is fed back to the participants of the design activity via email and if the result is negative, the design activity is restarted so that the design can be modified and resubmitted. Note that in the triggers, the names *AProposal* and *AReview* denotes the activity objects, which are instances of classes *CDocument* and *CReview* respectively.

```
class CDocument    { ... } //details omitted
class CReview      { ... } //details omitted
process PReview startat AProposal
{   //Activity to produce a CDocument object
    public activity AProposal handle CDocument
    {
       trigger TDesignFinished as     //trigger header
          AReview.activity.Resume()      //action
       after submit activity;         //event
    }
 //activity to generate an CReview object
    protected activity AReview handle CReview
```

```
    {
       trigger TReivewFinished as
          email.SendMessage(AProposal.activity.GetParticipants(),
                            "Review Result for" + AProposal.GetName(),
                            AReview.GetReviewSummary() ), //action 1
       AReview.Approved() ? AProposal.SetApproval():
                            AProposal.activity.Resume()   //action 2
          after submit activity;
    }
}
```

For each process instance, the P runtime system maintains some structures to record the instance's runtime state and to execute triggers. One of them is the process object, an instance of the process'es supporting class, which contains two member variables for each activity, one for the activity object and the other for the activity instance. The first member variable is for the process instance. For example, for the process above, the definition of its supporting class is:

```
class PReview_class4process
{  //supporting class for process PReview
   public      CProcessInstance  process; //the process instance
public      CDocument         AProposal; //for activity object
public      CActivityInstance AProposal_activity;//activity inst
protected   CReview           AReview; //for activity object
protectedCActivityInstance AReview_activity;//activity inst
}
```

CProcessInstance and *CActivityInstance* are predefined classes for process and activity instances. We will describe them in more detail later in Section 3.4. The P runtime system allocates a process object, denoted as Θ, when an instance is initiated and sets the values for its fields when activities are created. To execute a trigger, the runtime system first adds to the supporting class the binary form of a temporary method, which contains a *return* statement, "**return** *ConditionExpr*;", where *ConditionExpr* is the trigger condition, executes it and gets the return value. If the return value is *true*, another temporary method that contains an expression statement, "*ActionExpr;*", where *ActionExpr* is the action expression, is added and executed.

3.2 Process Inheritance

In P, process inheritance is a mechanism that allows all definitions, i.e. *activities* and *local classes* and *processes*, of one process, which is called a *superprocess*, to be part of those of another, which is called a *subprocess*. In the subprocess, these definitions can be used in a way as if they were defined in the subprocess itself. At the same time, the subprocess can define new definitions and/or to customize some definitions in its superprocess/es to fulfill its own specific needs. In this way, a process definition can be reused without losing the capability to be customized. Currently, we support *single inheritance*, which means that a process definition can have only one direct superprocess.

Accordingly, the supporting class for the subprocess is a subclass of the supporting class for the superprocess. However, since the super class already has one member variable of class *CProcessInstance* for the process instance, the subclass will no longer has such a variable. Instead, the first item in the memory block allocated for the subclass now contains the identifier of the memory block allocated for the superclass. In this way, the memory blocks for process objects are chained together. For example, suppose *PPaperReview* is a subprocess of *PReview*. The structure of the memory block of the process object for an instance of *PPaperReview* is shown in Fig.2, where a black cell stores a value converted from the pointer to either an activity or a process instance object (a C++ object in the memory space of the runtime system). Note that the first item of $\Theta_{PReview}$ now stores the converted pointer to process instance for *PPaperReview* (the conversion is done by changing the highest bit from 0 to 1, see Section 3.3 for details.)

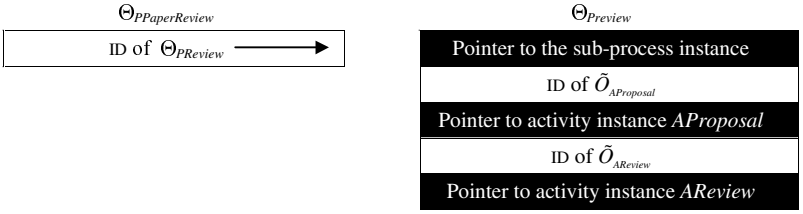

Fig. 2. Memory structure for a process object

3.2.1 Customization

An important purpose of using inheritance is to customize some aspects of superprocesses so that the process definitions can be more appropriate for a more specific situation. In P, we can customize a superprocess in a number of ways, e.g. *adding new activities, overriding or extending existing activities*. A critical issue is to organize the supporting structures for a subprocess so that all definitions in its superprocesses would function properly while instances of the subprocess are enacted.

3.2.1.1 Adding New Activities

A new activity, either a simple one or a nested process, can be added to a subprocess to handle a task that is not described in the superprocesses. By doing this, we customize the task decomposition of a process. The new activity can communicate with activities of superprocesses and of the subprocess itself. For each new activity in the subprocess, we need to add to its supporting class two new member variables and initialize the corresponding fields in the process object with appropriate values when the new activity is instantiated. If the new activity is a simple one, the first variable represents the activity object and the second variable represents the activity instance object. If the new activity is a nested process, the first variable represents the process object for the nested process and the second variable represents the process instance object.

The member variables in the subclass are totally indexed. Index of the first variable in a subclass equals the index of the last variable of its direct superclass plus one. For example, in Fig. 2, the index of *process*, the first variable of $\Theta_{PReview}$ is 1. If we add a

new activity in PPaperReview, the index of the first variable for this activity would be 6. In this way, any expression, no matter whether it is defined in a superprocess or in the subprocess, can be evaluated correctly with the process object for an instance of the subprocess.

3.2.1.2 Overriding or Extending Existing Activities

In some cases, we may just want to customize some aspects of an activity of a superprocess. For example, we can change the class of the activity object so that the object produced in this activity can be more specific; we can disable or enable some of its triggers to turn off or on the communication channels between this activity and others; we can add new triggers to the activity to change the way it communicate with other activities; we can change its participant specification; we can even replace the activity definition with a new one.

As a simple example, the code below defines a subprocess, *PPaperReview*, of *PReview* defined in Section 3.1. We first create a new class *CPaper*, which inherits *CDocument*. Then we customize the design activity by changing the class of the activity object to *CPaper*. The review activity is customized by specifying the participants to be a set of reviewers whose expertise matches the keywords of the submitted paper and by adding a new trigger that will send a message to the perspective reviewers every time the review activity is resumed. As indicated by the keyword *extending*, the two activity definitions in the subprocess customize the activities in the superprocess by extending them. Therefore, the trigger definitions in the superprocess will still be in effect.

```
class CPaper extend CDocument { … } //details omitted
process PPaperReview extend PReview
{   //make the activity object more specific
    extending activity AProposal handle CPaper
    {
    }
 //add participant specification and a new trigger
    extending activity AReview handle CReview
    {
       users reviewerdb.GetReviewers(AProposal.GetKeywords());
       trigger TNotifyReviewers as      //trigger header
          email.SendMessage(
             reviewerdb.GetReviewers(AProposal.GetKeywords()),
             "Paper for your review",
             AProposal.GetPaperURL())   //action
          after resume activity;        //event
    }
}
```

Overriding or extending an activity does not introduce a new activity (task) to the process. This means that for that activity, only one activity object and only one activity instance object will be created. For example, *PPaperReview* has only two activities, *AProposal*, which produces a *CPaper* object and *AReview*, which produces a *AReview* object. The P runtime system properly reorganizes the supporting

structures for a process instance to reflect this fact. For example, for an instance of process *PPaperReview*, after its two activities are both activated, the state of its process object is shown in Fig. 3. Note that $\Theta_{PPaperReview}$ has four variables for the two extending activities. However, the value of each variable of $\Theta_{PReview}$ will be set to that of the corresponding variable of $\Theta_{PPaperReview}$. In this way, the corresponding variables represent the same entity (an activity object or an activity instance object).

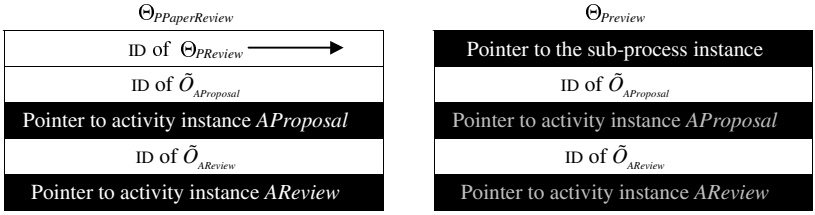

Fig. 3. The process object of process *PPaperReview*

Overriding and extending differ in how they affect the runtime system in searching the triggers in activities with the same name in the process inheritance hierarchy. Starting from the leaf node, the searching ends at the first overriding activity. To ensure that trigger actions and conditions referring the activity can be correctly executed or evaluated, we require that the class type of the overriding or extending activity is the same as or a subclass of that of the activity being overridden or extended. For an activity that is a nested process, we only allow to override it by defining a new nested-process activity whose process type is a subprocess of that of the existing one.

3.2.2 Activity Accessibility

Process inheritance and process nesting raise the issue of activity accessibility, i.e. whether a subprocess or a nesting process can communicate with the activities in its superprocesses or the nested process. For example, a process programmer may want an activity to be used for a completely internal purpose and don't want any activity in a subprocess or a nesting process to communicate with it. P has a mechanism that is very similar to how some object-oriented languages restrict the visibility of class variables and methods. We define three accessibility modifiers, i.e. *private*, *protected*, and *public*. A private activity is only accessible in the triggers, participants, etc. of the activities in the same process definition. A protected activity can be accessed by the activities in the same process and its subprocesses. A public activity of a nested process can be accessed from within a nesting process. The P compiler checks the accessibility of activities when a P process program is compiled. It reports a compiler error if the rules are violated.

3.3 Native Methods/Objects for External Resources

P is an interpretive language. The P interpreter manages the allocation, release, and referencing of P objects used for various purposes during the enactment of instances

and interprets the binary process codes compiled from P source programs. P also supports a mechanism that allows the body of a method of a P class to be a native function residing in a dynamical link library (DLL) on Win32 platforms (our current implementation). We call it a *native method*. When a native method is called, the P interpreter calls the corresponding native function with all actual parameters. The return value will be copied to P's storage space. The native function can be written in any language. Our mechanism is very similar to Java's Native Interface (JNI).

As a more advanced form of the *native method* mechanism, another mechanism allows us to introduce a C++ object running in the memory space of the P runtime system into the object space of the P interpreter as a P object, which we call it a *native object*. The P interpreter will direct a call to a method of a native object to the corresponding method of the C++ object. For this purpose, we need to write a P class definition that services as an interface description of the C++ object. (The class *CProcessInstance* in Section 3.4.1 is an example.) The identifier of a native object is a value converted from the pointer of the corresponding C++ object by changing its highest bit from 0 to 1. (On Win32 platforms, a pointer in the user space is within range 0 ~ 2^{31} -1. Therefore, the identifier of a native object will be in range 2^{31} ~ 2^{32} - 1. The identifier of an ordinary P object allocated in P interpreter falls in range 0 ~ 2^{31} -1. Therefore, the P interpreter is able to tell whether an identifier represents an ordinary P object or a native object.)

The native method/object mechanisms offer a fully extensible and programmable way for accessing any external services from within P process programs. They are consistent with the object-oriented pattern for programming P classes. For a certain resources, all we need to do is to define as a P class an interface for it and implement the native functions or objects. Based on this mechanism, we have successfully developed some P classes, e.g. *CTime*, *CMath*, *CIO*, *CFileSystem, CEMailAgent*, and *CDatabase*, and the related native codes for *time conversion*, *mathematical calculation*, *input/output*, *accessing files, sending emails*, and *accessing a relational database*, with several hours work.

3.3.1 Global Objects

In some applications, some resources may be shared by many process instances. In these cases, we can install dynamically in a P runtime system objects representing these resources as global objects, which are accessible from within any process instance enacted in that runtime system. The P runtime system has the programming interface for dynamically install and uninstall global objects. A global object can be either a native object, or a normal P object whose definitions are given in the P language.

The processes *PReview* and *PPaperReview* contain examples on how to use global objects. There we use two global objects, *email,* which is an instance of *CEMailAgent* for sending Internet email messages, and *reviewerdb*, which is an instance a subclass of *CDatabase*. These objects should be properly installed when instances of *PReview* and *PPaperReview* are enacted.

3.4 Predefined Objects for Internal Services

The P runtime system allocates in its memory space a C++ object that records the runtime information of the process instance. It also allocates for each activity instance a C++ object representing the activity instance. For reflection purpose, it installs these C++ objects as native objects in the P interpreter. In a process program, these objects can be accessed via two predefined identifiers, i.e. *process, activity*, in the same way as ordinary P objects are accessed.

3.4.1 Process

process represents a process instance. It is a native object, which is mapped to a C++ object maintained for the instance in the P runtime system. *process* is an instance of class *CProcessInstance*, whose methods can be divided into four categories:

- *Retrieving Information* for retrieving certain information, e.g. *process name, display name, state code, start time,* and end time, that is determined at runtime.
- *Manipulating Instance* for changing forcefully, i.e. suspending, resuming, terminate, and aborting, the state of the process instance.
- *Modifying Definition* for modifying dynamically the process definition of this instance so that the instance can be adaptive to changing working setting. Details on this topic are available in a technical memo [*Yang 2002*].
- *Creating New Instance* for creating dynamically a new process instance to coordinate the accomplishment a new task. The newly created instance will be a top-level instance and enacted independently of the current one.

The definition of *CProcessInstance* is given below.

```
class CProcessInstance
{  //the constructor
   public          CProcessInstance(){ }
   //the name of the process definition
   public CString GetProcessName()     { }
   //return the user friendly name
   public CString GetInstanceName()    { }
   //return the state of this process instance
   public tiny     GetState()          { }
   //Get the start time of the process instance
   public int      GetStartTime()   { }
   //Get the end time of the process instance
   public int      GetEndTime()        { }
   //abort this process instance
   public boolean  Abort()             { }
   //terminate this process instance
   public boolean  Terminate()         { }
   //resume this process instance
   public boolean  Resume()            { }
   //suspend this process instance
   public boolean  Suspend()        { }
   //create a new process instance
```

```
    public boolean  CreateInstance(CString csProcessName,
        CString csDisplayName, CString csArguments)  { }
    //modify the definiton of this process
    public boolean  ModifyDefinition(
        tiny tAction, CString csObjectName,
        CString csSubObjectName, CString csCode)     { }
}
```

3.4.2 Activity

activity represents an activity instance in a process instance. It is also a native object, which is mapped to the corresponding C++ object in the P runtime system. The P interpreter records the value converted from the pointer to the C++ object in the process object for that process instance. *activity* is an instance of class *CActivityInstance*, which has methods for retrieving certain information about the instance and manipulating it. For simplicity, we will not list the code here. Both *PReview* and *PPaperReview* have examples on how *activity* can be used. (P compiler will convert a notation like *AReview.**activity*** into *AReview_**activity***, which is a member in the supporting class for the process.)

4 Comparisons with Related Work

We have outlined the architecture of a library for process programming and presented our implementation in the context of the P language and system. To the best of our knowledge, this is the first attempt to find a systematic solution for building reusable libraries for process programming. Though some ideas such as *inheritance*, *external resource integration*, and *reflection* are by no means new and have been exploited in some form in other work, a comparison with this related work would reveal that our implementation of these ideas are in some sense novel.

Several groups have reported building a process library for sharing and analyzing purposes. [*Gruhn et al. 1998*] presents the mechanisms on how to manage the storage of process definitions in both a relational *DBMS* and the main memory. These mechanisms aim at improved performance. The *Process Handbook* aims at a repository of business process descriptions to help people redesign existing and invent new organizational processes and to share ideas of organizational practices. It uses *inheritance* as a second dimension, in addition to *task decomposition*, to explore the similarities among different processes. The descriptions are made with natural English, thus the mechanisms like these in this paper for process enactment are out of their scope. In an similar effort, the *Process Asset Library* [*Hart et al. 1992*], the goal is to organize all assets, e.g. *generic process architectures*, *cycle models*, *process elements (or subprocesses or steps)*, of business processes into one easily available online database. Again the description, organization, and categorization of business processes are their main focus.

Some process models and languages such as *EPOS*, E^3, *CSPL*, and *GED* [*Katzenstein et al. 2000*] support inheritance. However, the inheritance is limited to describing in an object-oriented fashion various entities, e.g. *artifacts*, *roles*, *tasks*, involved in software processes. We have not yet seen any process language that

supports truly process level inheritance. Thus none of them has ever tried to develop related enactment mechanisms. [*Aalst et al. 2001*] defines formally four different types of inheritance of processes modeled with WF-net, a special kind of PetriNet. Their main purpose is to analyze and tackle problems related to transferring running workflow instances to a new model. We focus on mechanisms for enacting properly processes inherited from other processes.

External tool integration has been a major concern of nearly all process languages in software engineering. For those based on classical languages, such as Ada-based *APPL/A*, Prolog-based *Merlin*, it is possible to use the libraries of the base language directly to access any external resources. Therefore they don't need any special construction. For newly defined languages such as *SPADE-1*, *CSPL*, and *HSPF*, they need to introduce new construction for specifying external tools. For example, *CSPL* has a *tool* construct; an HSPF program can define a *tool* section to define tools used in it; *SPADE-1* uses more powerful tool integration facilities, such as *Sun Tooltalk*, *Microsoft OLE2*, and *DEC FUSE*; and E^3 uses *CORBA*. Our native method/object based mechanism allows external resources to be integrated virtually without any limitation, though similar mechanism has been used in other general-purpose languages like Java.

Accessing to internal services is a kind of reflection, a feature also available in other language like Smalltalk and Java. Reflection in certain limited forms is found in some process languages. For example, *HSPF* has a special data type *status* and two special operations *snap* and *resume* for recording and changing the enactment status of a process; *MELMAC* has a command for modifying a process model fragment immediately before it is enacted; *EPOS* and *SPADE-1* offers basic, reflexive constructs to support the specification of the "process of change" as part of the process itself. P's *process* and *activity* offer a more comprehensive reflection mechanism to use from within process programs all the enactment services provided by the P runtime system. Also it is extensible in the sense that we can easily add new functions. An added advantage is that they can be used in a consistent object-oriented manner.

5 Conclusions and Future Work

Process programming as a new programming paradigm is believed to be critical in many tightly related areas, e.g. software engineering, workflow management, and business management. We believe that a library with reusable elements for programming cooperative processes flexibly is essential for the acceptance, popularity, and success of this new paradigm. We defined an architecture for such a library and present an implementation in the context of a process programming language named P. We demonstrated that process inheritance can be an efficient vehicle for process reuse and developed the mechanisms that implement the inheritance semantics.

A public release of the P runtime system that contains all the work discussed here is available at http://blrc.edu.cn/research/p/. The system has been successfully used as the programming environment for a graduate level course on CSCW in Tsinghua University and as the software platform for several projects in e-commerce and workflow management. Recently, we have developed a process program for the

ISPW6 reference problem [*Kellner et al. 1991*]. These experiences show that the library concepts discussed here work pretty well.

However, there is still a long way to go. Making our library much richer, e.g. by adding more reusable process definitions into our library, by developing special-purposed libraries for different application areas, is in our near-term plan. The architecture and mechanisms need to be further explored and validated in a wider range of practical applications, with which we may learn something to improve and enhance our concepts and mechanisms.

A long-term work we are currently considering is whether it is necessary to define a standard binary format of process programs, and if the answer is yes, what format should we define. We believe that a standard format will promote significantly process sharing and reusing. This attempt conforms to NIST's goal on *Process Specification Language* [*http://ats.nist.gov/psl/*] and WfMC's on developing a common process interchange standard based on XML [*WfMC 2002*]. However, our effort differs from these in that it tries to achieve sharing from a much lower level.

The work itself is very challenging due to many factors, both technical and non-technical. For example, process programs are *executed* by process systems, which may adopts different process meta-models. It may be difficult, if not impossible, to map the concepts of one meta-model to the ones of another. However, if this can be done, all process systems will have a common ground to play on. The libraries for process programming may then be developed with different process languages.

Acknowledgements. The work presented here is based on the author's PhD work, which was supervised by Prof. Shi, Meilin at Tsinghua University and supported by several grants from China NSF and 863 Plan. The author is grateful for all the comments and suggestions from the students who ever used P for their coursework. The author wants to thank Alfred V. Aho and the anonymous reviewers for their insightful comments and suggestions.

References

Aalst, W. M. P. and Basten, T. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 2001, 270(12):125–203

Ambriola, V., Conradi, R., and Fuggetta, A. Assessing process-centered software engineering environments. *ACM TOSEM*, 1997, 6(3):283–328

Bandinelli, S., Nitto, E. D., and Fuggetta, A. Supporting cooperation in the SPADE-1 environment. *IEEE TOSE*, 1996, 22(12):841–865

Chen, J. Y. CSPL:an Ada95-like, unix-based process environment. *IEEE TOSE*, 1997, 23(3):171–184

Conradi, R., Hsgaseth, M., Larsen, J.-O., *et al*. EPOS: Object-oriented cooperative process modeling. In: *Software Process Modeling and Technology*, Finkelstein, A., Kramer, J. and Nuseibeh, B. *Eds*. Research Studies Press, London, U.K., 1994

Cortes, M. and Mishra, P. DCWPL: A Programming Language for Describing Collaborative Work. In: *Proc of ACM CSCW*, Cambridge, 1996, 21–29

Curtis, B., Kellner, M. I., and Over, J. Process modeling. *CACM*, 1992, 35(9):75–90

Fuggetta, A. Software process:a roadmap. In: *Proc of ACM/IEEE ICSE*, Limerick, 2000, 27–34

Gruhn, V. and Schneider, M. Workflow management based on process model repositories. In: *Proc of ACM/IEEE ICSE*, Kyoto, 1998, 379–388

Gruhn,V. and Jegelka, R. An evaluation of FUNSOFT nets. In: *Proc of the 2nd European Workshop on Software Process Technology*, 1992, LNCS. Springer-Verlag, New York.

Hart, H., Doland, J., Drake, D., *et al*. STARS process concepts summary. In: *Proc of ACM Conf on TRI-Ada*, Orlando, 1992 , 570–594

Jaccheri, M. L., Picco, G. P., and Lago, P. Eliciting software process models with the E^3 language. *ACM TOSEM*, 1998, 7(4):368–410

Katayama, T. A hierarchical and functional software process description and its enaction. In: *Proc of ACM ICSE*, Pittsburgh, 1989, 243–252

Katayama, T. and Motizuki, S. What has been learned from applying a formal process model to a real process. In: *Proc of IEEE ISPW*, Washington D.C., 1991, 79–81

Katzenstein, G. and Lerch, F. J. Beneath the surface of organizational processes: a social representation framework for business process redesign. *ACM **TOIS***, 2000, 18(4):383–422

Kellner, M. I., Feiler, P. H., Finkelstein, A., *et al*. ISPW-6 software process example. In: *Proc of IEEE ISPW*, Redondo Beach, 1991, 176–186

Klein, M., Dellarocas, C., and Bernstein, A. *eds*. *Proc of ACM CSCW Workshop on Adaptive Workflow Systems*, Seattle, 1998. (http://ccs.mit.edu/klein/cscw98/.)

Krueger, C. W. Software reuse. *ACM Computing Surveys*, 1992, 24(2):131–183

Malon, T. W., Crowston, K., Lee J., *et al*. Tools for inventing organizations: toward a handbook of organizational processes. *Management Science*, 1999, 45(3):425–443

Osterweil, L. Software processes are software too. In: *Proc of ACM/IEEE ICSE*, Monterey, 1987, 2–13

Peuschel, B. and Schafer, W. Concepts and implementation of a rule-based process engine. In: *Proc of ACM/IEEE ICSE*, Melbourne(AU), 1992, 262–279

Salimifard, K. and Wright, M. Petri net-based modelling of workflow systems: an overview. *European Journal of Operational Research*, 2001, 134, 664–676

Sutton, S. M., Jr., Heimbigner, D., and Osterweil, L. J. APPL/A: a language for software process programming. *ACM TOSEM*, 1996, 4(3):221–286

WfMC. Workflow process definition interface – XML process definition language. *WFMC-TC-1025*, 2002. (http://www.wfmc.org/.)

Yang, G. and Shi, M. *Cova*: a programming language for cooperative applications. *Science in China, Series F*, 2001, 44(1):73–80

Yang, G. Inheritance-based dynamic process modifications. *Bell-Labs Technical Memo*, Nov. 2002

Generating a Process Model from a Process Audit Log

Mati Golani and Shlomit S. Pinter

IBM Research Laboratory in Haifa, Haifa 31905, Israel
{matig, shlomit}@il.ibm.com
http://www-suif.stanford.edu/~shlomit/

Classification: Process mining

Abstract. Workflow systems utilize a process model for managing business processes. The model is typically a directed graph annotated with activity names. We view the execution of an activity as a time interval, and present two new algorithms for synthesizing process models from sets of systems' executions (audit log). A model graph generated by each of the algorithms for a process captures all its executions and dependencies that are present in the log, and preserves existing parallelism.
We compare the model graphs synthesized by our algorithms to those of [1] by running them on simulated data. We observe that our graphs are more faithful in the sense that the number of excess and missing edges is consistently smaller and it depends on the size and quality of the log. In other words, we show that our time interval approach permits reconstruction of more accurate workflow model graphs from a log.

1 Introduction

Constructing business processes is a central issue for companies [2,3]. Managing processes in an automatic or semi-automatic fashion result in a significant reduction of cost and improves efficiency of business operations, thus, enabling fast adaptation to changing requirements and more. As a result, developing techniques for constructing and managing business processes is an active research area [1,4].

Workflow systems utilize a visual model of information flow that is used for monitoring and managing systems that execute actions (also called activities or tasks) of predefined situations. The actions together with constraints on execution order between them define the business process [5]. Commercial workflow systems and management consoles need a model of the business process for scheduling agents (e.g., computers) to execute the actions, control production, etc. (see [5,2]). For modeling a business process, most ERP/CRM products use embedded workflow model [6,7].

Many organizations that run their systems using legacy applications do not have a model of the processes within the organizations. Current tools for model

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 136–151, 2003.

detection operate on the resource level only. Thus, there is a need for tools to build business process level models, especially when all executive level measures such as "return on investment", or SLA quality are derived from this level rather than the resource level. There are few methods for constructing a business process model from information stored in a workflow log (a collection of process execution logs). We follow the approach that represents the model as a directed graph (workflow graph) with nodes representing activities and an edge from node A to node B represents that there is a process execution in which A must finish executing before B starts. In practice, a single business process model can permit an execution that include a given activity and another execution that may not include it. Thus, for each process execution the participating edges are selected with a Boolean function associated with each edge. The function determines whether the control flow or not along that edge.

Another paradigm is to deal with workflow evolution that updates the process models according to the logs (see [8,6,9,10,8,11,12,13]).

Contribution: Unlike the event based models in which the execution of an activity is represented as a single event, we view the execution of an activity as a time interval (life span) based on its starting and ending events that are present in the workflow log. For this view we can recognize concurrent activities with a single process execution, since intersecting life span intervals in a process execution represent concurrent activities. In event based models concurrent activities cannot be recognized in a single process execution. In addition, based on the data in a log of some workflow systems we can recognize conditions that specify when activities are immediately concurrent (i.e., modeled as AND-join from some activity in the model). We present a new algorithm based on our interval model. Given a workflow log we show that compared with the event based approach, the interval approach permit the reconstruction of a more accurate process model graph.

We define causality dependence between activities with respect to the process execution log and describe an algorithm that, given a process execution log, generates a workflow graph that guarantees the following:

- *Completeness:* Every process execution in the log can be generated from the workflow graph.
- *Correctness:* All the dependencies with respect to the execution log exist in the workflow graph.
- *Preserving parallelism:* If two activities, A and B, are concurrent with respect to the log then there are two paths from start to end such that one includes A and does not include B and the other includes B and does not include A.

A workflow graph generated by our algorithm resemble the original workflow graph (the input for the system that generated the log) by having a minimum number of excess and missing edges. This minimum depends on the size and quality of the log. The time complexity of the algorithm is $O(|ex||V|^2)$, where $|ex|$, $|V|$ are the number of executions and activities in the log, respectively.

In their seminal paper [1] the authors build a workflow graph by considering each activity as an atomic event. This was achieved by taking the finish activity

event to be a node in the graph. We show that when the interval view is taken there is more information that can be used for reconstructing a more accurate model. We compare our results with those of [1] and show that the quality of the generated models is better with our algorithm.

Background and Related work: The topic of process discovery has been dealt for some time [14,4,15,16,17,18]. Our work can be viewed as an extension to the event based approach taken by [1,16]. They use a single event to denote the execution of an activity and reconstruct a directed acyclic graphs as the model. The common approach as describes in those studies was to identify the different business processes in the execution log and gather the participating activities in each execution.

In [14,4] Cook and Wolf searched for software processes by investigating events in concurrent environment. They present three methods for process discovery: A neural networks based, algorithmic approach which builds a finite state machine where states are fused if their futures (in terms of possible behavior in the next k steps) are identical, and a Markovian approach which uses a mixture of algorithmic and statistical methods and takes noise in consideration. The Authors propose specific metrics (entropy, event type counts, periodicity, and causality) and use these metrics to discover models out of event streams. However, they do not provide an approach to generate explicit process models. In [15] they provide a measure to quantify discrepancies between a process model and the actual behavior as registered using event-based data.

The model generated by [14,4,19] are Petri-nets and finite automata. For the reconstruction of the models they use frequencies of events sequences in the log. A learning method for process mining is presented in [20]. A comprehensive survey on process mining techniques is presented in [21].

The remainder of this paper is organized as follows: In Section 2, we describe process model and introduce our interval model. In Section 3 we present our new algorithms for reconstructing a workflow graph from rich executions log. Then, in Section 4, we present our simulation environment followed by experiment results and discussion in Section 5. Finally, in Section 6, we conclude with a short summary.

2 Process Model

The common approach to view workflow modeling is to use a directed graph composed of a set of nodes each labeled by a name of an activity (two of the labels are start and end), and a set of directed edges with a Boolean control function associated with each one of them. There is an edge from a node labeled by activity A to a node labeled by activity B only if there is an execution where B can start executing immediately after the termination of activity A; in such a case the control function on the edge evaluates to one (TRUE). We say that B is a *successor* of A and A is a *predecessor* of B if there is an edge from A to B. The decision whether B must execute following the execution of A depends on

the value of the control function applied to some data available when A is done. For more details on the model see [1,16].

In this approach, the execution of an activity is thought to be atomic (instantaneous) and no two activities may ever start at the same time; a process execution in this model is a list of activities. We latter on extend the model to capture a more accurate view in which the execution of an activity is an interval along the time axis and two intervals may intersect.

Initially, we assume that there are no directed cycles in the graph and every label appears at most once in each execution. The assumptions can be removed by re-labeling activities.

A *legal flow* is a maximal connected subgraph of the workflow model graph such that the control function is evaluated to one on each edge in the subgraph (all nodes are of AND type), both the start and the end activities are in the subgraph, and every activity (node) is on a directed path from start to end. A legal flow graph, over a set of activities, is a partial order representing all possible ways to schedule the selected activities (all possible executions). The meaning of the order is that an activity can be executed only if all of its predecessor activities in the flow graph finished executing (AND join) and only when its done, its successor activities can start executing (AND split). Since the successor activities can execute concurrently we say that they are *immediately concurrent*.

We define an *execution* over a workflow graph to be a consistent linearization of a legal flow (i.e., preserves edge ordering) that is represented by a list of activities, $A = a_1, a_2, ..., a_n$, starting with the start activity, a_1, and ends with the end (target) activity, a_n.

From the assumptions on the workflow graph we know that no activity appears more than once in an execution list.

Figure 1 is an example of a workflow model graph and two legal flow subgraphs. In flow (a) the control function is evaluated to 0 on the edge from A to C and the edge from B to D. In flow (b) the value is 0 on the edge from B to C. Execution (A, B, C, D) is a legal execution in both flows, but (A, C, B, D) is an execution only in Flow (b). Those are also the only executions legal in our example.

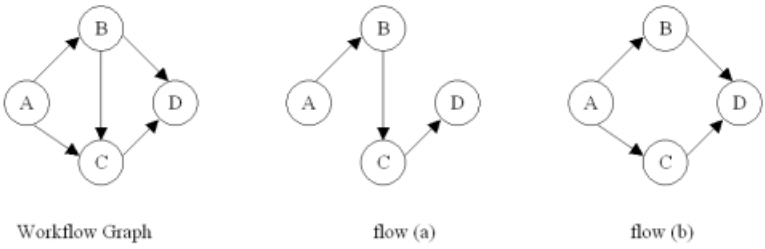

Fig. 1. A sample workflow model graph and two legal flows

For a given workflow model graph we define two conditions:

[CA] Activities a_i and a_j are *concurrent activities* if there are two executions of a legal flow such that a_i appears before a_j in one execution, and a_i appears after a_j in the other execution.

[NS] Activity a_i is not *im-successor* of a_j if $a_i \neq a_{j+1}$ in all legal executions. From the edge semantics it means that a_i never immediately succeeds a_j.

Note that concurrent activities are defined with respect to a single flow. Both conditions will be refined for the interval view.

In Figure 1 B and C are concurrent activities due to flow (b) for which both executions are legal, and D is not im-successor of A.

There is no way to reconstruct the workflow model graph from the two executions since they are also the only executions legal for a workflow model graph for which there is no edge from B to C. The task of reconstruction is harder when only some of the executions are known. Assume that we know only of (A, B, C, D), then we can only reconstruct a strait line graph. We later on show that in the interval model we get, in addition, executions of the form (A, B $\|$ C, D) indicating that B and C were executed concurrently. In such a case we can infer the existence of flow (b) with a single execution. Graph (b) can't replace (a) as the model since (a) also comprise the option for the control function from B to D to evaluate to 0.

Given two activities a_i and a_j in a workflow graph. Activity a_i *depends* on activity a_j iff whenever a_i appears in some legal execution over the workflow graph, a_j must appear in that same execution some time earlier (i.e., $j < i$ and the time of the termination event of a_j is smaller than the time of the ready event of a_i). This definition of dependence implies that a_i cannot run unless a_j ran sometime earlier (causality). In Figure 1 activity D depends on A.

For example: Given a workflow graph (e.g., for a process of verifying car building) that contains the activities: "ship a car" and "final checks", we know that "ship a car" depends on "final checks". Indeed, our definition implies that in every execution of the flow graph it must be the case that if "ship a car" appears then "final checks" must appear sometime earlier.

Claim. : Given a workflow model graph and two of its activities a_i and a_j, such that a_i depends on a_j, then there is a path from activity a_j to activity a_i in the graph and in every legal flow.

2.1 Log Structure and the Interval Model

The log of a workflow contains monitored data that refers to process executions. Each execution of a process is composed of executed activities ordered on a time axis. In some systems the monitored data is a single event per executed activity and the time in which it occurs. Some other systems generate richer audit trails that contain more events per executed activity. One example is the MQWorkflow Audit trail that follows the WFMC audit standard [22] and provides logs that contain start and stop events for each activity (in addition to other events),

the details of the running business process instance, and its business process template.

For each process instance (execution) we consider those logged records that contain the following events of an activity: {ready, started, restarted, ended normally, force finished, terminated}. Clearly, one can remove those executions that contain noisy events; e.g., if a force finished event indicates noise in the process workflow the corresponding executions can be removed. We say that an event is a *start* event if it is one of {started, restarted}, and it is a *termination* event if it is not a ready or a start event. Each record in the log contains additional data such as: time, process name, process ID (an execution), Activity name, Activity ID, and User ID.

An activity has to be ready before it can start. Once it has started it can end normally, be forced, or terminate. The *life-span* of an executed activity is defined to be the time interval from its start event to its termination event, and the *extended_life-span* is defined to be the time interval from its ready event to its termination event. Each event indicates a change in the state of the system; thus, for example, a system in a ready state will stay in that state until a start event occurs.

In what follows we refer only to events of a single process ID and view it as an execution. In a distributed system the time stamps of events are given locally by each component. As a result we may have a case were the order between two events, as indicated by the time stamps, is incorrect. We assume that the time stamps of events imply that if activity B is a successor of activity A (there is an edge from A to B in the graph) then the time of the ready event of activity B is greater than the time of the termination event of A. This does not imply a total order on the events, but rather it says that the clocks keep causality order. The assumption on the time stamps can be easily maintained either with a global clock when exists, or by adding the time of the termination event to the data used in the control functions that trigger the successor's activities.

Since errors, like exceptions, may create execution instances which are not compatible with the business process, the respective log have a noisy data. Such noise is handled during the construction of the work flow graph.

Note that the successor relationship between activities is kept both by the times of a richer audit trail and the execution generated from a log that has a single event per executed activity.

The following condition can be derived from the information in a rich audit trail in addition to CA and NS above:

[rCA] For a given list of activity events (execution) of some process, activities a_i and a_j are *concurrent activities* if there is an overlap between the extended_life-spans of a_i and a_j (e.g. there is a time slot in which both activities are in a ready/active state).

3 Reconstructing a Workflow Graph

We next describe the problem of reconstructing a model workflow graph form a given set of executions (process log) that are generated in a workflow system.

Given two activities a_i and a_j in a process log. *Activity a_i depends on a_j with respect to the process log* iff whenever a_i appears in some execution in the log, a_j must appear in that execution some time earlier and the time of the termination event of a_j is smaller than the time of the ready event of a_i.

Note that the definition of dependence over the process log in [1] does not imply our notion of dependence over the log or even over the workflow graph since they, for example, permit the execution of the dependent activity without its cause.

Since many legal executions may not be present in the log and since parallel activities appear sequentially in an execution, some none dependent activities can be considered as dependent with respect to the log.

For a given process log L we define:

[CAL] Activities a_i and a_j are *concurrent activities with respect to L* if one of the following conditions is satisfied:
- There are two executions in L, over the same set of activities, such that a_i appears before a_j in one execution and a_i appears after a_j in the other execution.
- If there is an execution in L such that the extended_life-spans of a_i and a_j overlap.

[NSL] Activity a_i is not a successor of a_j with respect to L if for every execution in L at most one of a_j or a_i is present.

A reconstructed workflow graph G is *consistent* with a process log L if the following conditions hold:

- *Completeness:* Every execution in L can be generated from G.
- *Correctness:* All the dependencies with respect to L exist in G; i.e., For every dependence with respect to L, there is a corresponding path in G.
- *Preserving parallelism:* If two activities, A and B, are concurrent with respect to L then there are two paths, in G, from start to end such that one includes A and does not include B and the other includes B and does not include A.

A consistent workflow graph of a given process log is *optimal* if it has the smallest number of excess and absent edges (with respect to the original model graph) over all consistent workflow graphs defined by that log.

3.1 Constructing the Process Execution Graph

We present an algorithm for reconstructing a workflow graph with the interval approach.

The workflow model graph is generated by combining graphs that are generated from the process executions in the log. In the first step, a graph is generated

for each execution (execution graph). The interleaving of intervals in a single execution makes this graph non-trivial. During this step we keep information of overlap intervals. Next, a single graph is generated for all the execution graphs that have the same set of activities. In the last step the graphs are merged together and strongly connected components are handled.

Combining executions graphs is done in two steps. The reason for combining first executions that run over the same set of activities stems from the observation that, in general, they were all run on a single subset of the workflow graph (i.e., the parameters for the business process that used to select the participating edges were the same). Thus, the resulting graph corresponds to a flow graph.

Given an execution from a process log where activity events are sorted based on time, the following algorithm builds an execution graph. During the generation of the graph we maintain two sets of nodes: `current frontier` and `next frontier`. The nodes in the `current frontier` are the latest nodes that were added to the graph and their out degree is zero at that stage. In addition, we maintain two markers along the time axis: `current time`, and `next time`.

Algorithm for generating an execution graph

The first node in the graph is the start activity (the first in every execution). Let the start node be the `current frontier` set and the time of its finish event be the `current time`.

The following steps are executed until the time of the finish event of the end (target) activity is the value of `current time`.

1. Let the time of the first finish event that followed `current time` be `next time`.
2. Add to the graph a node for each activity that has its ready event between `current time` and `next time`. This set of nodes is the `new frontier`.
3. Add an edge from each node in `current frontier` to every node in `new frontier`.
4. Change the `current frontier` to be the set of activities that include the activity finished in `next time` and all the activities that finished between `next time` and the first ready event that followed `next time` (when exists). Let `current time` be the time of the latest finish event among the activities in this set.

We demonstrate the algorithm on the graph in Figure 2.

The following is a set of rich executions where a letter corresponds to a ready event and a tagged letter is a termination event:

1. (A,A',B,C,D,C',E,B',F,D',F',G,G',E',H,H')
2. (A,A',C,B,D,D', B',G,C',E,F,F',G',E',H,H')
3. (A,A',C,C',F,E,E',F',H,H')
4. (A,A',D,D',G,G',H,H')

In Figure 3 and Figure 4 we provide the process execution graphs of the four executions.

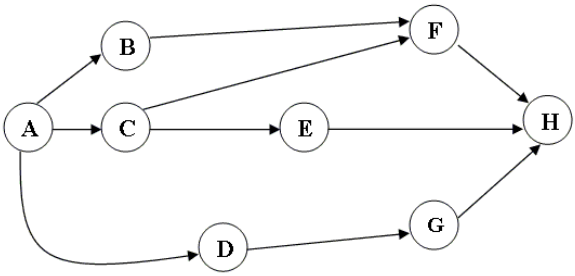

Fig. 2. A sample work flow graph

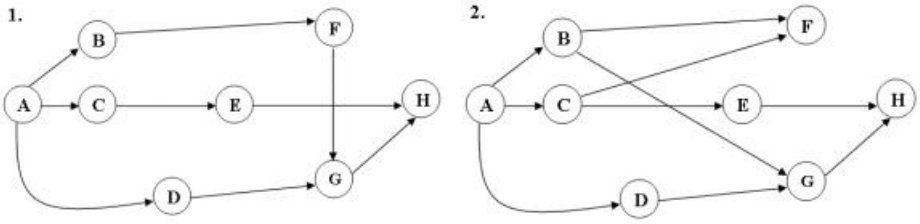

Fig. 3. Execution graphs for executions 1 and 2

It is simple to show that an execution graph is a legal flow graph for that execution. Nevertheless, from a single execution we may not be able to find all the available parallelism. In such a case, at this stage, we may have redundant edges that may be removed when combining the graph with other execution graphs on the same set of activities.

To handle noise we maintain a weight on each edge. The weight indicates how many executions are coherent with the edge. Thus, assuming that noise is relatively rare (e.g. less than 5%) edges with small weight can be filtered out.

3.2 Combining Execution Graphs

In this step we first combine execution graphs that have the same set of activities. We refer to the new graph as the reconstructed flow graph.

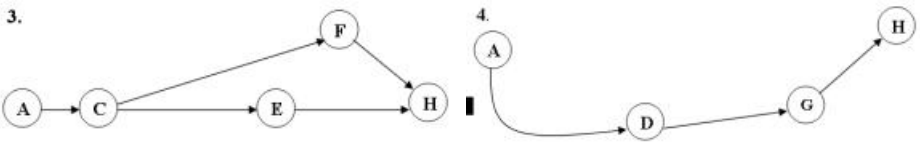

Fig. 4. Execution graphs for executions 3 and 4

During the scan of the events (in the step above) we also generated a set, `fedges`, of pairs that comprise the "forbidden" edges. If two activities overlap then there is a pair (forbidden edge) in this set. For example, in execution 1 we can see that activity B overlap each of C, D, and E.

For a given set of execution graphs over the same set of nodes, the *reconstructed flow graph* is a graph $F = (V, E)$ such that:

- V is the set of activities comprising the nodes of the execution graphs.
- The set of edges E is generated by taking the union of all the edges in the execution graphs and removing the forbidden edges.

The type of concurrency presented by a forbidden edge is always true since it is derived from a single execution. Other forms of concurrency are derived from multiple executions.

To show that all the executions can still be generated from the flow graph we use two observations: (i) by the assumption that all the executions over the same set of edges were generated from a single flow graph (same set of parameters - values of the control functions - were used to select the graph), all the combined executions are consistent with a single partial order, thus there are no cycles; (ii) if an edge was removed then there is an execution - the one that is responsible for getting the edge into the forbidden set - that explore the relevant parallelism and thus its continuation can be used to compensate for the removed edge.

If the log does not contain all the possible executions over a set of activities, then the set of edges in the reconstructed flow graph may be different from that of the corresponding original legal flow graph in spite of them being compatible (both can be used for generating the given set of executions). Note that at this stage all the nodes are of type AND as in the corresponding legal flow graph.

Next we merge all flow graphs to get the *merged flow graph*. Following this merge, some of the nodes are changed to have an OR semantics. The merge is done similar to the previous step but the forbidden set is now empty. Condition NSL and the second part of condition CAL are kept during the merge but this may not be the case with the first part of CAL. As a result we may get cycles in the merged flow graph. In the last step we break the strongly connected components in the merged flow graph and get the *workflow graph*.

For a given graph G, we denote by $N(G)$ the set of nodes in G, and $A(G)$ is the set of its edges.

Algorithm for breaking strongly connected components
Given a merged flow graph $F = (V, E)$.

1. The nodes in each strongly connected component H in F, are partitioned into 4 sets:
   $M_h = \{v \in N(H) \mid A(v) \subseteq A(H)\}$ that is, all there adjecent edges are in the strongly connected component.
   $B_h = \{v \in N(H) \mid exist\ x, y \in \{N(F) - N(H)\}\ and\ (v, x), (y, v) \in A(v)\}$ that is, they have at least one incoming edge and one outgoing edge that are not in the strongly connected component.

$I_h = \{v \in N(H) - N(B_h) \mid exists\ x \in \{N(F) - N(H)\}\ and\ (x, v) \in A(v)\}$
$O_h = \{v \in N(H) - N(B_h) \mid exists\ x \in \{N(F) - N(H)\}\ and\ (v, x) \in A(v)\}$
(i) Remove the edges in $A(B_h)$ and the edges going from $N(B_h)$ to $N(I_h)$
(ii) Remove the edges going from $N(M_h)$ to $N(I_h)$

2. The procedure is applied recursively to each of the remainig strongly connected subcomponents.

Claim. : Every execution in the log can be generated from the merged flow graph.

In our running example, the resulting flow graphs of executions 1 and 2 are given in Figure 5. We can see that the edge from F to G of the first execution is not in the combined graph. The graph does not change when the other two flow graphs are merged. The only difference is that some of the nodes are now have an OR semantics. It is simple to see that the graph is compatible with all the executions and only a single edge (the dashed edge) is not in the original workflow graph.

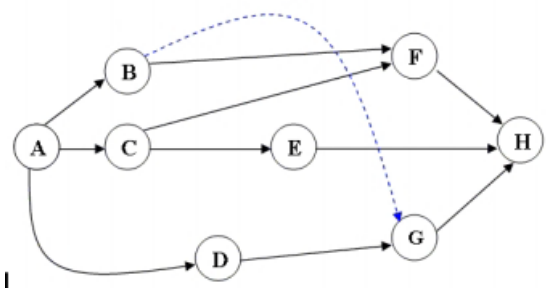

Fig. 5. The resulting flow graph

4 Simulation Environment and Algorithms

For comparing different algorithms we wrote a process graph generator and used it to generate a process graph and a set of legal flow subgraphs. Executions were generated from the flow graphs to serve as input to the reconstruction algorithms.

4.1 Generating a Process Graph

Business processes do not take an arbitrary shape. They are usually more structured than random graphs; we use that knowledge to guide some decisions when generating workflow graphs for testing our algorithms. Every graph is a DAG with one source and one sink. Every node has at most 4 adjacent edges. This

forces an out/in degree of 3 or less to each node. The edges are generated by randomly selecting a node, u, and then a second node, v. If there is an edge from u to v we remove it, otherwise we add an edge from u to v only if it complies with the restriction on process graphs (e.g., the in/out degree is smaller then 3, there is no path from v to u). The number of times that this step is carried on is bounded by a number proportional to a polynomial in the number of nodes (the type of the graph). To complete the process graph, the source and target nodes are added with edges from the source node to nodes with in-degree zero and edges from nodes with out degree zero to the target node.

We use the process graph generator to generate three types of process graphs: with 10, 25, and 50 nodes (activities). For the experiment we used 50 graphs from each of the three types to get a collection of 150 process graphs.

For each one of the 150 process graphs we generated a set of legal flow graphs. The number of flow graphs generated depends on the type of the graph, such that 5, 10, and 10 flow graphs are generated for process graphs of size 10, 25, and 50 nodes respectively.

4.2 Generating a Flow Graph for the Simulation

A legal flow graph is generated from each process graph with the following procedure:

1. Put the source node in a working set.
2. Select a node, v, from the working set.
3. Randomly select an outgoing edge from v and mark it. The rest of the outgoing edges from v are randomly chosen (with probability 0.4) and the chosen edges are marked. This guarantees that there is at least one outgoing edge from v which is marked.
4. The respective destinashion nodes of marked edges are being added to the working set if they were never there before.
5. Delete from the graph outgoing edges from v that were not marked, and remove v from the working set.
6. Recursively remove from the graph nodes that have no incoming edge (other then the source node).
7. Repeat this procedure from Step 2 until the working set is empty.

The motivation for selecting an additional outgoing edge with probability of 0.4 is implied from the observation that many real life flow graphs have out degree of two. As a result our simulated graphs are similar to real life scenarios.

Due to the probabilistic nature of the procedure for generating flow graphs, some of the nodes and edges of the simulated process graph are never selected. Thus, we take the simulated process model graph to be the union of all the flow graphs that are generated from a single process graph. This process model graph is now fully covered by the flow graphs and it is used for comparing the quality of the reconstruction algorithms.

For each flow graph we generate an execution log that contains random executions (lists of events). The number of executions per flow graph varied from 100 to 150000. See technical report for more detailes.

5 Experiment Results

We tested three algorithms. Our interval algorithm (Interval Sorted) described in section 3.1, a modified version (Interval) of our algorithm in which we merge all the execution graphs into one graph in a single step, and the Non-Interval algorithm, which is the second algorithm described in [1]. For the last algorithm we use the finish events to represent activities executions.

Every value presented in the following histograms is the average over 50 runs of each algorithm. The quality of the algorithm is measured by comparing the synthesized graphs with the generated process graph (the input). The edges that are in the generated graph but not in the synthesized graph are the missing edges, and those in the synthesized graph but not in the generated graph are the excess edges.

We show the average percentage of missing and excess edges for each of the three algorithms as function of the size of the log.

In Figure 6 each input graph has 10 nodes.

The average missing edges percentage of each of the three algorithms is relatively stable and is independent with the log size. The Interval Sorted algorithm gives the best results. The average excess edges percentage is relatively stable for the Non-Interval algorithm, and varies from 4% down to 2%. The interval algorithms are much more sensitive and vary from 1% in short logs down to 0.1% in long logs.

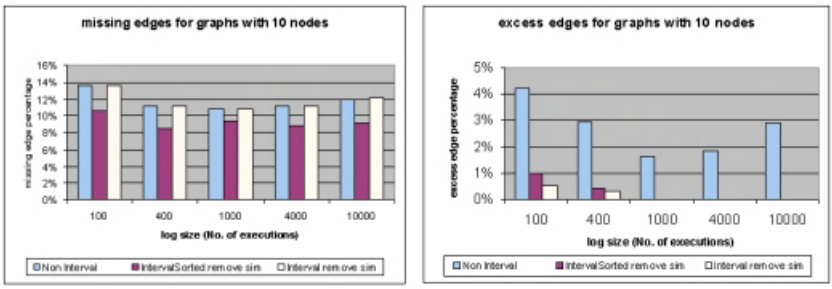

Fig. 6. Comparing the algorithms on logs from graphs with 10 nodes

In Figure 7 each input graph has 17 nodes.

The average missing edges percentage of each of the three algorithms is relatively stable and is independent with the log size. The Interval Sorted algorithm gives the best results (between 10% and 9%). The average excess edges percentage is relatively stable for the Non-Interval algorithm. As for 10 nodes graphs, the interval algorithms are more sensitive to the size of the logs.

In Figure 8 each input graph has 25 nodes.

The average missing edges percentage and the average excess edges are relatively stable for the Non-Interval algorithm. The Interval Sorted is sensitive to

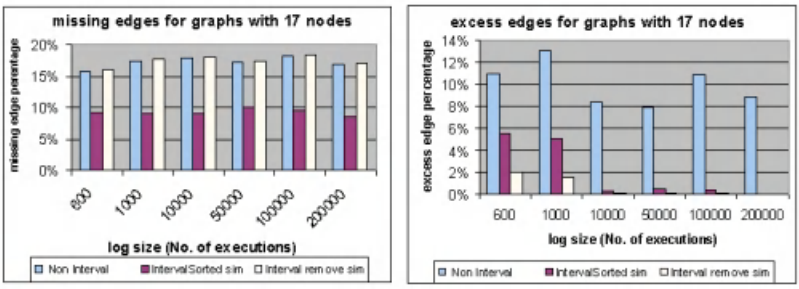

Fig. 7. Comparing the algorithms on logs from graphs with 17 nodes

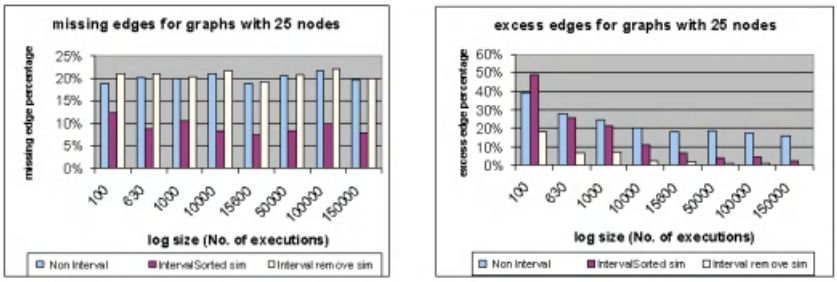

Fig. 8. Comparing the algorithms on logs from graphs with 25 nodes

the size of the log and improves when the logs have more executions; it provides the best results when the number of missing edges is counted. The average missing edges percentage is relatively stable for the Interval algorithm, yet the algorithm is sensitive to the log-size for the excess edges. The interval algorithm gives the best results for excess edges.

In Figure 9 we compare execution times versus log size, and versus the size of the input graph.

5.1 Summary

One issue of concern is the percentage of missing edges. All three algorithms miss some portion of the original edges. This portion does not depend on the number of executions (size of the log). There is a threshold of about 10% missed edges even in very large logs. An experiment that builds the graphs from the logs without any cleaning showed that there are still about the same 10% of missing edges (the number of excess edges was very high).

Both interval algorithms are more accurate (in resemblance to the original model) than the Non-Interval. The policy of what algorithm to use is naturally a question of what is more important.

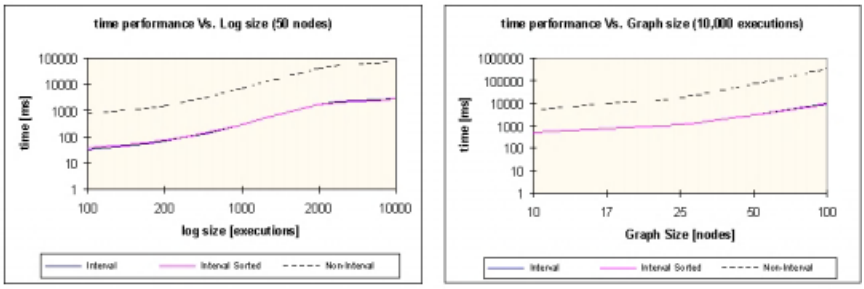

Fig. 9. Performance results

References

1. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings. Volume 1377 of Lecture Notes in Computer Science., Springer (1998)
2. Georgakopoulos, D., Hornick, M.: An overview of workflow management: From process modeling to workflow authomation infrastructure. Distributed and Parallel Database **3** (1995) 119–153
3. Wastell, D., White, P., Kawalek, P.: A methodology for business process re-design: experiences and issues. Journal of Strategic Information Systems **3** (1994) 23–40
4. Cook, J.E., Wolf, A.L.: Event-based detection of concurrency. In: Sixth International Symposium on the Foundation of Software Engineering (FSE-6). (1998) 35–45
5. Hollingsworth, D.: The workflow reference model. Technical Report TC00-1003 issue 1.1, workflow management coalition, UK (1995)
6. Ellis, C., Kkeddara, K.: A workflow change is a workflow. In: W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors. Business Process Management: Models, Techniques, and Empirical Studies, volume 1806 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2000, Berlin, Germany, Springer-Verlag (2002) 45–63
7. van der Aalst, W., van Hee, K.: workflow Management: Models, Methods, and Systems. Number TC00-1003 issue 1.1. MIT press, Cambridge, MA (2002)
8. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. In: In proceedings of ER 96, Cottubus, Germany (1996) 438–455
9. van der Aalst, W., Jablonski, S.: Dealing with workflow change: Identification of issues and solutions. International Journal of Computer Systems, Science,and Engineering **15** (2000) 267–276
10. Agostini, A., Michelis, G.D.: Improving flexibility of workflow management systems. In: In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, Lecture Notes in Computer Science. Volume 1806., Berlin, Germany, Springer-Verlag (2000) 218–234
11. Klein, M., Dellarocas, C., A. Bernstein, e.: Towards adaptive workflow systems. In: Proceedings of the CSCW-98 Workshop Towards Adaptive workflow Systems, Seattle, Washington (1998)

12. Klein, M., Dellarocas, C., editors, A.B.: A knowledge-based approach to handling exceptions in workflow systems. Special issue of the journal of Computer Supported Cooperative Work **9** (2000) 399–412
13. Schimm, G.: Process miner - a tool for mining process schemes from event-based data. In: Proceedings of the 8th European Conference on Artificial Intelligence (JELIA), volume 2424 of Lecture Notes in Computer Science, Berlin, Springer-Verlag (2002)
14. Cook, J.E., Wolf, A.L.: Discovering models of software processes from event-data. ACM, Transactions on software Engineering and Methodology (1998) 215–249
15. Cook, J., Wolf, A.: Software process validation: Quantitatively measuring the correspondence of a process to a model. ACM Transactions on Software Engineering and Methodology **8** (1999) 147–176
16. Herbst, J., Karagiannis, D.: An inductive approach to the acquisition and adaptation of workflow models. In: Proceedings of the IJCAI'99 Workshop on intelligent Workflow and Process Management: The new frontier for AI in Business, Stockholm, Sweden (1999) 52–57
17. Herbst, J., Karagiannis, D.: Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. International Journal of Intelligent Systems in Accounting, Finance and Management **9** (2000) 67–92
18. Weijters, A., van der Aalst, W.: Rediscovering workflow models from event-based data. In: Proceedings of the 11th Dutch-Belgian Conference on Machine Learning. (2001) 93–100
19. van der Aalst, W., van Dongen, B.: Discovering workflow performance models from timed logs. In: In Y. Han, S. Tai, and D. Wikarski, editors, International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002), volume 2480 of Lecture Notes in Computer Science, Berlin, Germany, Springer-Verlag (2002) 201–217
20. Maruster, L., Weijters, A., van der Aalst, W., van den Bosch, A.: Process mining: Discovering direct successors in process logs. In: Proceedings of the 5th International Conference on Discovery Science (Discovery Science 2002), volume 2534 of Lecture Notes in Artificial Intelligence, Berlin, Germany, Springer-Verlag (2002) 364–373
21. van der Aalst, W., Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow mining: A survey of issues and approaches. Technical report, workflow management coalition, UK (2003)
22. workflow management coalition: Interface 5 - audit data specification. Technical Report WFMC-TC-1015 issue 1.1, workflow management coalition, UK (1998)

Contracting Workflows and Protocol Patterns

Andries van Dijk

Deloitte & Touche Management & ICT Consultants, Postbus 300,
1180 AH, Amstelveen, The Netherlands
anvandijk@deloitte.nl

Abstract. Inter-organizational business processes often involve contracting. ICT solutions for contracting processes must offer high flexibility in changing the structure of the contracting process. This can be achieved by 'process-aware' software components which are configured by an explicit model of the contracting process: the contracting workflow. However, the design of a contracting workflow from scratch is a complex task. We propose a solution in which contracting workflows are composed from standard building blocks and show that protocol patterns for business transaction protocols are a necessity for making these standard building blocks available. Finally, we propose a number of protocol patterns for the negotiation phase in a transaction.

1 Electronic Contracting

Inter-organizational business processes often involve contracting. When one organization buys something from another organization, a distinction between 'products' and 'services' is often made. Although products and services differ in many ways, the question is whether these differences are relevant from the perspective of the contracting process. This question is answered by for instance Normann and Ramirez [7], who state "whether customers buy a 'product' or a 'service', they really buy access to resources". Hence, the authors use the term 'offering' to refer to both 'product' and 'service'. Others, like Merz et al [6], have the same approach when they consider payments and tangible goods as services too. In this paper, we will use the term 'service' as a synonym for both 'product' and 'service'.

Service contracting involves information exchange between partners, for which electronic communication is one of the options. The term 'electronic contracting' was already mentioned by Lee in 1988 [5]. In this paper, we define a 'contract' as 'an agreement between two parties in which the mutual obligations are stated'. Furthermore, we define the term 'electronic contracting' as 'a contracting process in which the communication between parties is performed by electronic means and in which the processes at the involved parties are supported by computer applications.' The term 'electronic contracting' is used for a variety of phenomena. This paper is focused on a

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 152–167, 2003.

specific part of this area, which is demarcated by the following characteristics that define a class of service contracting processes.

- *Loosely coupled organizations*
  We assume a relationship between service clients and service providers where all communication is performed by exchanging structured messages, of which only the data types (static aspects) and constraints on the sequence of message types (dynamic aspects) are mutually agreed. We assume no knowledge of each others business processes for the participating organizations.

- *Buyer side only*
  Electronic contracting of services always involves a buyer (client) and a seller (provider). Although these parties communicate via a common message protocol, they execute different processes. This paper focuses on the contracting workflow executed by the buyer (service client). The seller (service provider) is treated as a black box, of which only the external interface (transaction protocol) is known.

- *Multiple required services, Multiple available providers*
  A contracting process is performed for a business case in the enterprise information system. This paper focuses on the more complex contracting processes where each business case requires N different services to be contracted, for which M different service providers are available.

- *Dependencies between services*
  We assume dependencies between required services that define the order in which services must be contracted. For example, service B must be contracted when service A has been completed (sequential relation). Or, service B must be contracted only if service A could not be contracted (alternative relation).

Apart from a demarcation of the class of processes under consideration, we further limit the scope of this paper by focusing on the dynamic aspects of contracting processes only. Data aspects involved in contracting processes, for instance deriving the details of required services from case data or evaluating a received offer, are not in the scope of this paper. A framework for capturing the data aspects of contracting processes is given by Van Dijk [3].

2 Modeling Technique and Approach

Different approaches have been proposed for modeling of the communication between partners in a buying process. In this paper, we view contracting processes as *inter-organizational workflows* and use *Petri Nets* as modeling technique. This choice is made for the following reasons.

- Workflow management techniques based on Petri Nets have a sound theoretical basis and have been successfully applied to internal business processes like shown by Van der Aalst and Van Hee [1]. Since business processes are becoming inter-organizational increasingly, the application of workflow management techniques to inter-organizational processes is an obvious choice.

- Workflow management techniques have proven to be a good solution for repeating, well-structured and potentially long-running processes. The character of the demarcated class of service contracting processes has many similarities with this kind of processes.

- Workflow management techniques are increasingly integrated in software tools for electronic messaging. Apparently, the market recognizes the usefulness of workflow management in combination with electronic business.

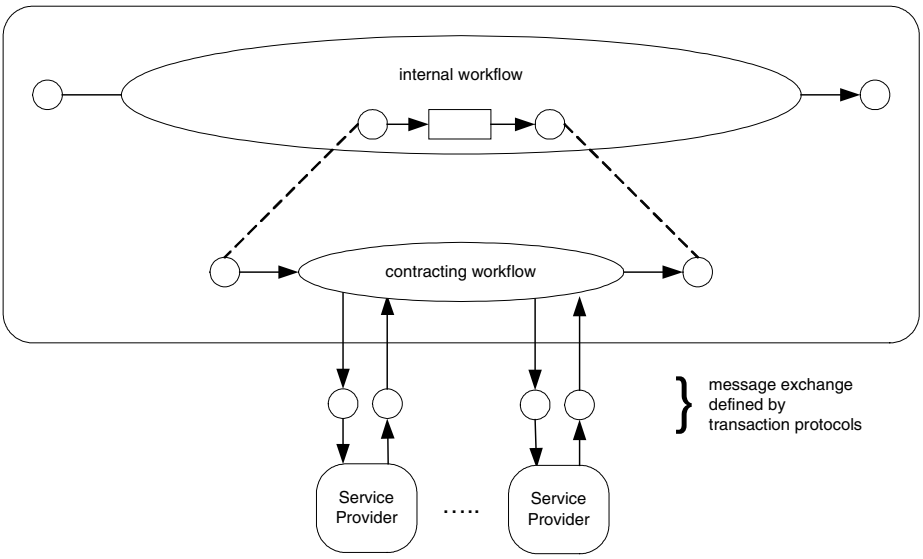

Fig. 1. Position of Contracting Workflow and Transaction Protocols

Clearly, a contracting workflow that involves multiple required services, each of which is controlled by a transaction protocol with multiple messages, can grow to a complexity where it is very difficult for a user to design this workflow from scratch. We therefore propose the following approach.

1. We define a standard high-level workflow structure for the contracting process of a single service (see section 3). The tasks in this high-level workflow need further refinement by replacing it by a sub-net. We assume a library of sub-nets that can be used for that purpose.

2. We start with an empty workflow (start and end place only) and add the standard high-level workflow structure for each different service involved in the process.

3. We add transitions and places to the high-level workflow to model the dependencies between the services (contracting requirements).

4. We add transitions and places to make the high-level workflow sound.

5. We create the final contracting workflow by substituting each task in the high-level contracting workflow with a proper sub-net from the library of sub-nets..

3 A Framework for Contracting a Single Service

A number of frameworks for contracting processes, based on buying products or services from a third party, can be found in literature, e.g. Action Workflow, DEMO (Dynamic Essential Modeling of Organizations) by Dietz [2] and BAT (Business as Action game Theory) by Goldkuhl [2]. These frameworks share the idea that business transactions consist of four phases:

- **Phase 1: Specification**
  In the specification phase, the service client specifies the details of the service to be contracted. In fact, because each service requires a service provider to execute his business process, the specification phase is in its essence the creation of a case token for the workflow in the service providers information system.

- **Phase 2: Negotiation**
  The negotiation phase aims at establishing a contract with a service provider for the specified service. This research focuses on service contracting processes in situations of partial knowledge. This, and the fact that external service providers are often autonomous organizations, implies that a service client can not simply *assign* a task to a service provider but has to *negotiate* with the service provider instead. A contract is established only if there is an offer made by the provider and an acceptance of the offer by the client. The negotiation phase ends either with a contract after which the execution phase starts, or without contract after which the process ends (failed).

- **Phase 3: Execution**
  If a negotiation process resulted in a contract, both service client and service provider will have to fulfill the commitments they entered in the contract. An important aspect of the execution phase is the exchange of status information from service provider to service client, used by the service client to monitor the fulfillment of the contract. The execution phase ends either with the completion of the execu-

tion after which the acceptance phase starts, or it ends with an abortion of the execution after which the process ends.

- **Phase 4: Acceptance**
  The objective of the acceptance phase is to obtain a mutual agreement on the fulfillment of commitments. The service provider declares the fulfillment of his commitments, the service client accepts this declaration and settles the financial obligations towards the service provider. Settlement of financial obligations is however outside the scope of this research. During the acceptance phase, information must be exchanged between service client and service provider. At this point, mutual satisfaction is obtained and the transaction is completed.

This results in a standard structure for contracting one service:

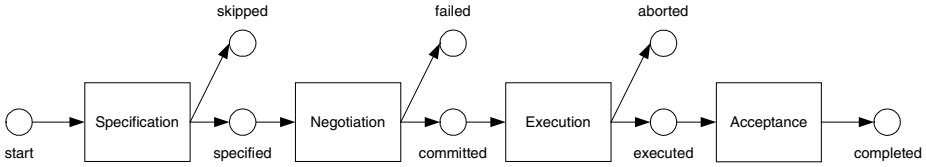

Fig. 2. High-level workflow structure for contracting a single service

4 A Contracting Workflow for Multiple Services

When multiple services are involved in a contracting process, there are always contracting requirements defining the dependencies between the different services. For example: service A can only be contracted after service B has been contracted. Or: when service A can not be contracted, service B must be contracted. This section discusses the rules according to which a contracting workflow for multiple services is composed from the building blocks defined before and from the contracting requirements.

Copy the high-level workflow structure
The starting point for each contracting workflow is a source place ‘start’ and a sink place ‘end’. The first step in creating the contracting workflow is to add the transitions and places (see Figure 2) of the high-level workflow structure for each individual service. This step can be fully automated in the configuration environment of a software component for contracting processes, when the user defines a list of required services.

Model the dependencies between services
The second step in creating the contracting workflow is to add transitions and places that model the contracting requirements, i.e. the conditions under which a token is

produced in the 'start' place of the structure from Figure 2. This step can be partially automated when we assume a relative small number of types of dependencies between services. If the user can define the contracting requirements by selecting from the list of services and a list of dependency types, the corresponding changes to the contracting workflow can be made automatically. To illustrate this, we give two examples of dependencies between services and the corresponding representation in the contracting workflow.

Example 1 – 'B starts when A failed'
In this type of dependency, service B is an alternative for service A. The dependency is modeled by an extra processor that consumes a token from place 'failed_A' and produces the token in place 'start_B'.

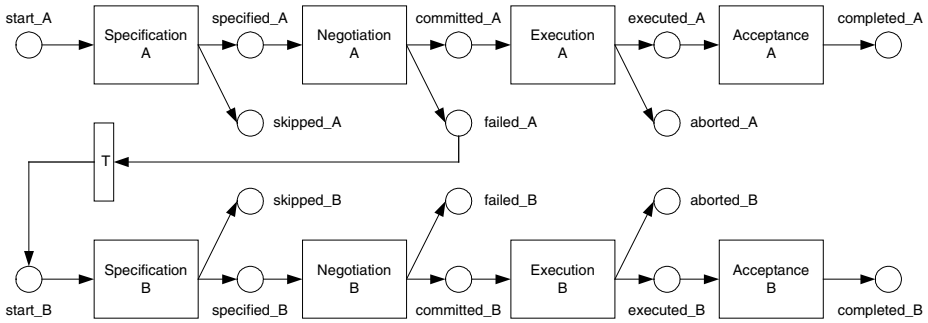

Fig. 3. Example of dependency 'B starts when A failed'

Example 2 – B starts when A is committed
Another typical type of triggering is when the contracting for a service B is started when another service A has been contracted. This dependency is modeled by duplicating the place 'committed_A' into 'committed_1A' and 'committed_2A' and adding a transition that consumes a token from place 'committed_1A' and producing the token in places 'committed_2A' and 'start_B'.

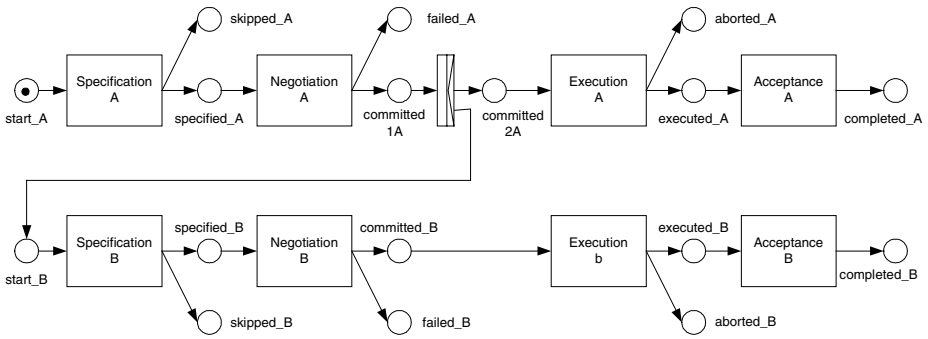

Fig. 4. Example of dependency 'B starts when A is committed'

Make the high-level contracting workflow sound
We require contracting workflows to be a sound WF-net if we omit the places via which message tokens are exchanged with workflows of service providers. At this point, we have created a structure which is not a sound WF-net. Therefore, the third step in creating the contracting workflow is to add those transitions and places to the high-level contracting workflow that make the resulting high-level contracting workflow a sound WF-net. This can be done by analyzing the distribution of tokens in possible end-states. There after, new transitions and places are added in such a way that in each end-state the tokens that define the end-state are consumed and one token is produced in place 'end'. This task can be automated completely. An example of a sound high-level contracting workflow is given in Figure 5. The example is about a company in which employees have to travel frequently. Each business trip requires two flights to be booked (outbound and inbound). If the employee is not able to travel on one day, a hotel reservation has to be made. Finally, if the employee wants to, a rental car must be made available at the airport of arrival.

Refinement of the high-level contracting workflow
The high-level contracting workflow that we have defined so far is a sound workflow, but is still a high-level workflow where the tasks need further refinement by replacing each task with a sub-net. The structure of the sub-net by which a task in the high-level workflow is replaced depends highly on the transaction protocol that defines the dynamics of the information exchange between service client and service provider. A standard set of sub-nets that can be substituted in the task of the high-level workflow can only exist when there is some type of standardization in transaction protocols. This is why we propose to standardize a relative small number of transaction protocol patterns, on the basis of which we can define the business transactions in specific situations.

5 Transaction Protocol Patterns

Clearly, there is not a single transaction protocol common to all possible service types. Differences in transaction protocols are likely to occur due to differences in legislation, business model, fulfillment processes, etc. However, although we can not present a single transaction protocol for all services, we are able to define *patterns* for transaction protocols. We define a protocol pattern as "a transaction protocol pattern captures the underlying common structure of a set of transaction protocols with different message types but identical dynamic behavior." Since a transaction protocol encompasses the consecutive negotiation, execution and acceptance phases, it can be seen as composed of three smaller transaction protocols, one for each phase. In the rest of this section, we will present patterns for the negotiation phase. Requirements to protocol patterns and correctness criteria can be found in Van Dijk [3].

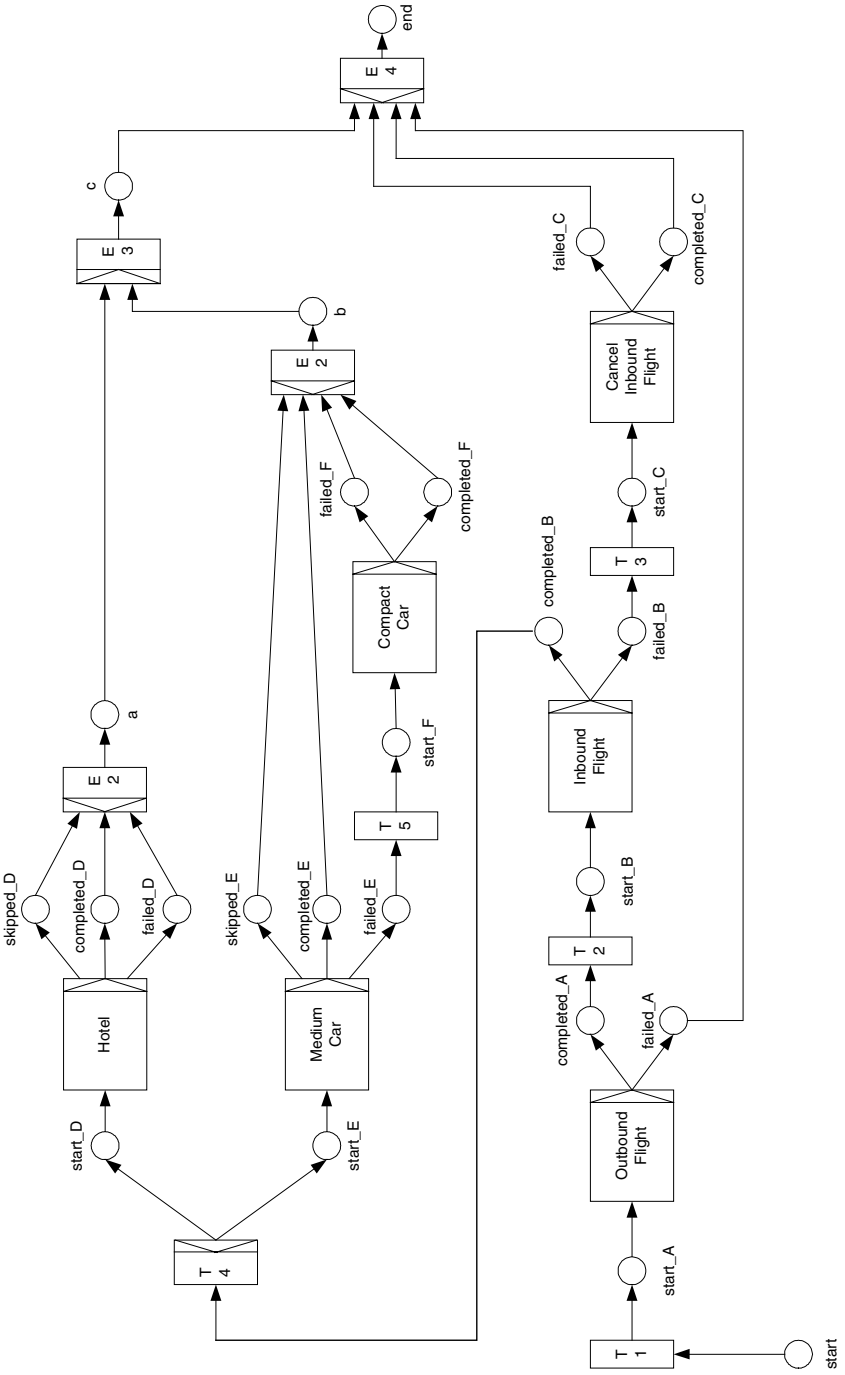

Fig. 5. Example of a sound high-level contracting workflow

Negotiation Pattern: 'Implicit Accept'

This negotiation pattern is used in situations where there is no explicit response by the service provider to a request made by the service client. Instead, the contract is considered to be established after the request has been made. Clearly, this variant can only be applied under circumstances where the implicit accept is agreed in previous agreements or laws.

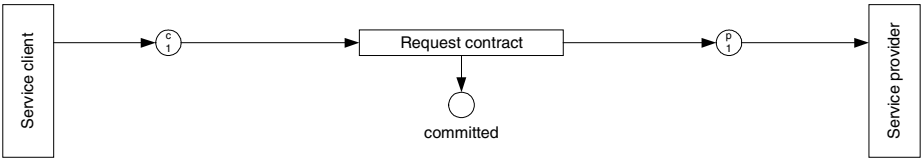

Fig. 6. The 'implicit accept' protocol pattern

Negotiation Pattern: 'Binding Request'

This negotiation pattern is used in a situation where a service client makes a binding request to a service provider, who responds by either accepting or rejecting the request. If the service provider accepts the request, a service contract is established, after which the execution protocol starts. If the service provider rejects the request, neither of the parties has a commitment to each other and the transaction ends.

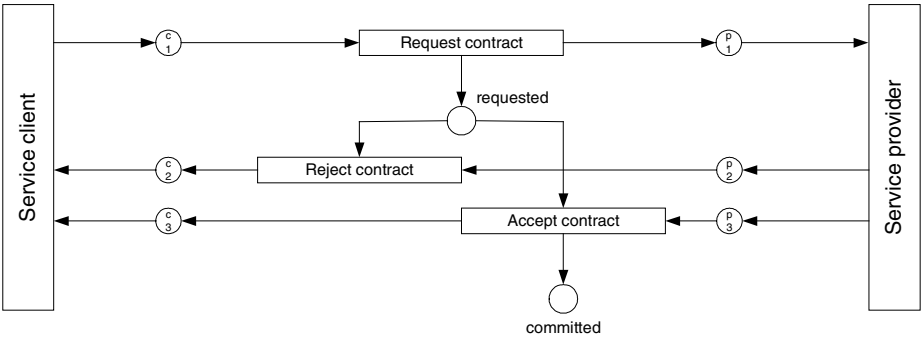

Fig. 7. The 'binding request' protocol pattern

Negotiation Pattern: 'Single Binding Offer'

Instead of requesting a contract from a service provider directly, a service client can also request an offer from a service provider. Offers can be binding or non-binding. This negotiation pattern is based on a single binding offer given by the service provider to the service client. When the service provider receives a request for an offer, he either responds by sending a notification that he will not make an offer (e.g. because he is not able to fulfill the request) or he responds by sending an offer message. When the service client receives an offer, he will either accept the offer after which a service contract is established and the execution phase starts, or he rejects the offer after which neither of the parties has a commitment to each other and the transaction ends.

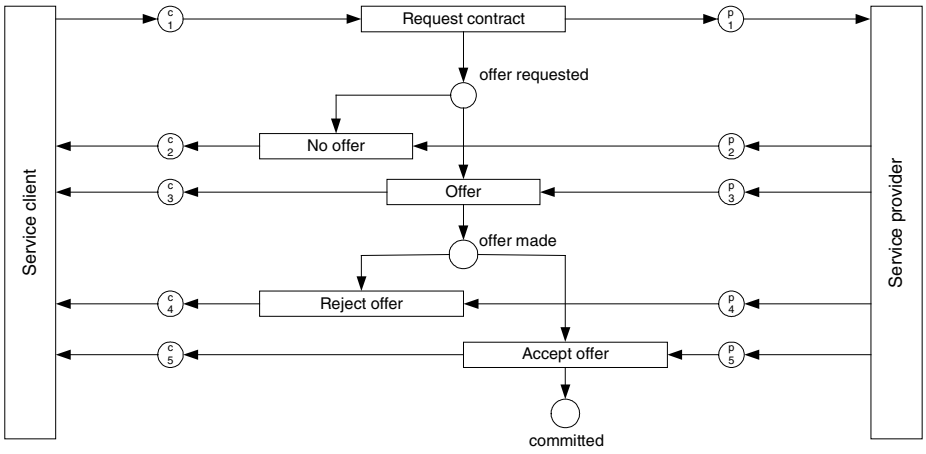

Fig. 8. The 'single binding offer' protocol pattern

Negotiation Pattern: 'Single Non-binding Offer'

An extension to the 'single binding offer' pattern emerges when the service provider sends a non-binding offer instead of a binding offer. This leaves the possibility that after the service client accepted the offer the contract can still not be established, e.g. because the resources required for the fulfillment have been exhausted in the period between sending the offer and accepting it. The pattern is equal to the 'single binding offer' pattern, but has two additional message types that can be received by the service client after he accepted the offer. The confirm accept message indicates that a service contract has been established and the execution phase started. The reject accept message indicates that no contract could be established after all which ends the transaction and leaves both parties without any obligation towards each other.

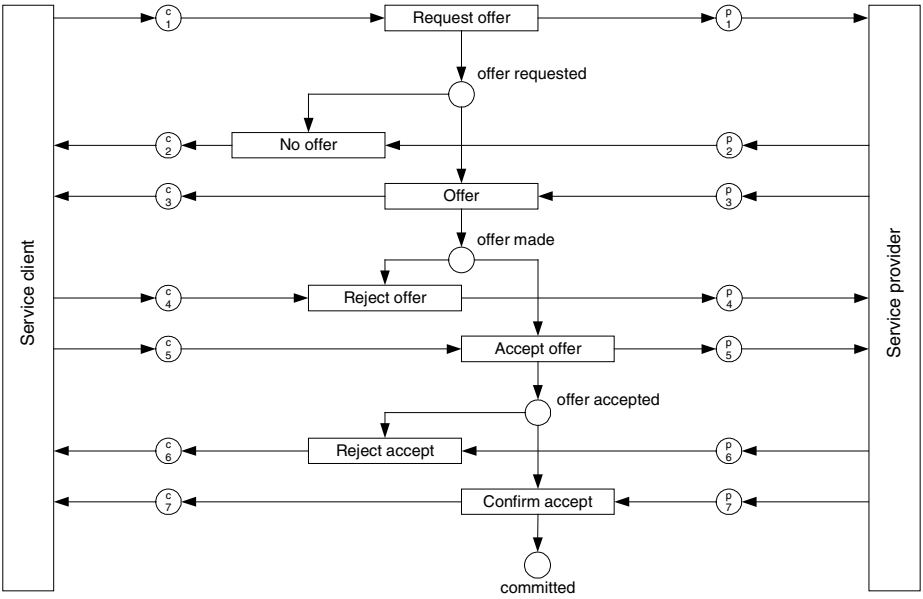

Fig. 9. The 'single non-binding offer' protocol pattern

Negotiation pattern 'Multiple Binding Offers'

An extension to the 'single binding offer' pattern is to allow the service provider to send multiple binding offers instead of a single binding offer, in order to give the service client alternatives to choose from. The negotiation pattern starts with a request from the service client to the service provider. The provider answers either by sending an offer, or by sending a notification that he will not send an offer. If the service provider sends one offer, he can send an arbitrary number of additional offers thereafter. When the service client received one or more offers, he evaluates them and either rejects all offers by sending a reject offers message or he accepts the offer he finds 'best' by sending an accept offer message, which contains a reference to the particular offer he is accepting. If the service client accepts an offer, a service contract is established and the execution phase starts. Otherwise, the transaction ends and leaves both parties without any obligation towards each other.

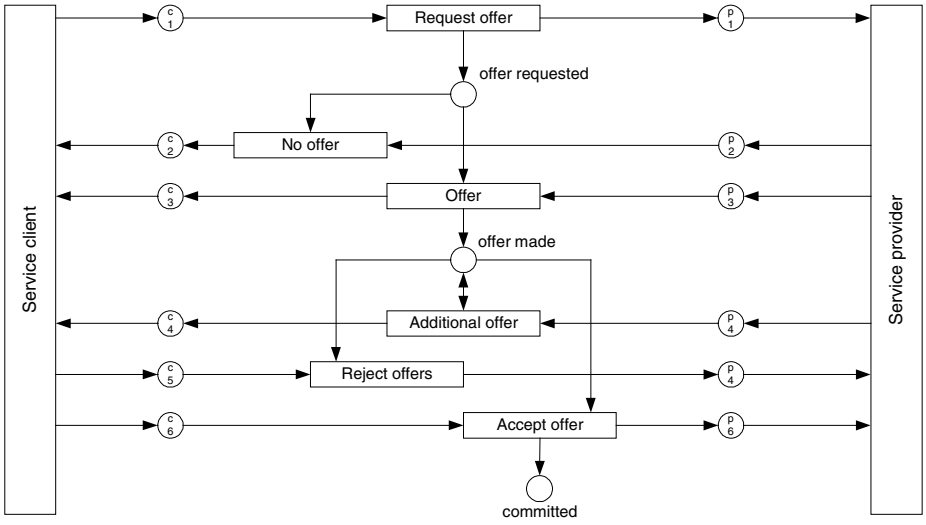

Fig. 10. The 'multiple binding offers' protocol pattern

Negotiation pattern: 'Multiple Non-binding Offers'

An extension to the 'multiple binding offers' pattern emerges when the service provider sends non-binding offers instead of binding offers. This leaves the possibility that after the service client accepted an offer the contract can still not be established, e.g. because the resources required for the fulfillment have been exhausted in the period between sending the offer and accepting it. The pattern is equal to the 'multiple binding offers' pattern, but has two additional message types that can be received by the service client after he accepted the offer. The confirm accept message indicates that a service contract has been established and the execution phase started. The reject accept message indicates that no contract could be established. However, the transaction returns to state 'offer made', in which the service client can cancel the negotiation or accept another offer from the pool of offers received from the service provider. In the same state, the service provider can send new offers to the service client.

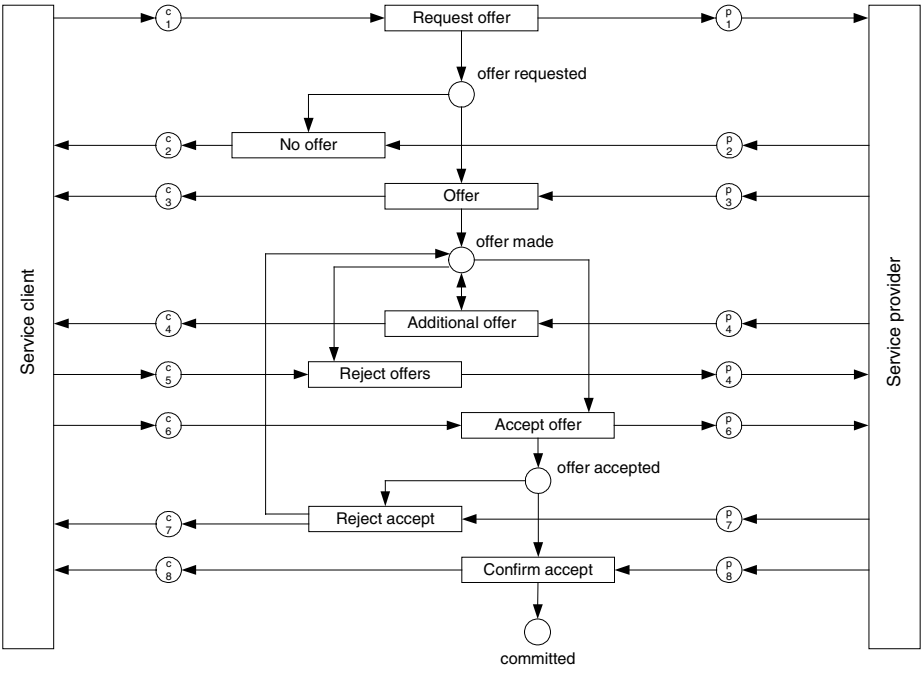

Fig. 11. The 'multiple non-binding offers' protocol pattern

Negotiation Pattern: 'Single Binding Counter Offer'

An extension to the 'binding request' pattern is to allow the possibility of a binding counter offer by the service provider as a third type of response to a direct request for a contract. When the service provider makes a counter offer, the negotiation process enters a state in which the service client can either accept or reject the counter offer and in which the service provider can withdraw the counter offer. If the service client accepts the counter offer, a service contract is established and the execution phase starts. Otherwise, the transaction ends leaving both parties without any obligations towards each other.

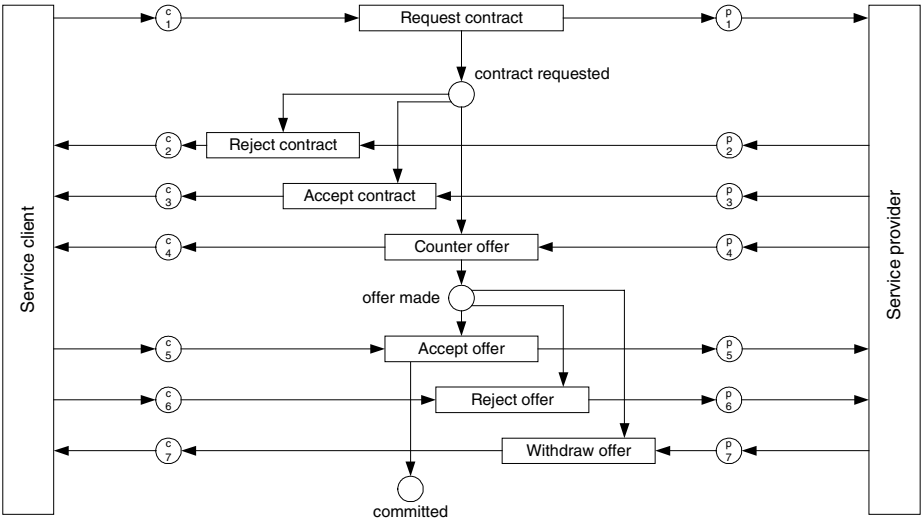

Fig. 12. The 'single binding counter offer' protocol pattern

Negotiation Pattern: 'Alternating Binding Counter Offers'

An extension to the 'single binding counter offer' pattern is to allow the possibility of a counter offer to be followed by a different counter offer made by either the service client of service provider. If a counter offer is made, it replaces all earlier made counter offers. Hence, a maximum of one counter offer can be under consideration at each moment. The party that made the current counter offer can replace it by a different counter offer or withdraw it. The party that did not make the counter offer under consideration can either accept it, reject it, or make a counter offer himself. If a party accepts an offer made by the other party, a service contract is established and the execution phase starts. Otherwise, the transaction ends leaving both parties without any obligations towards each other.

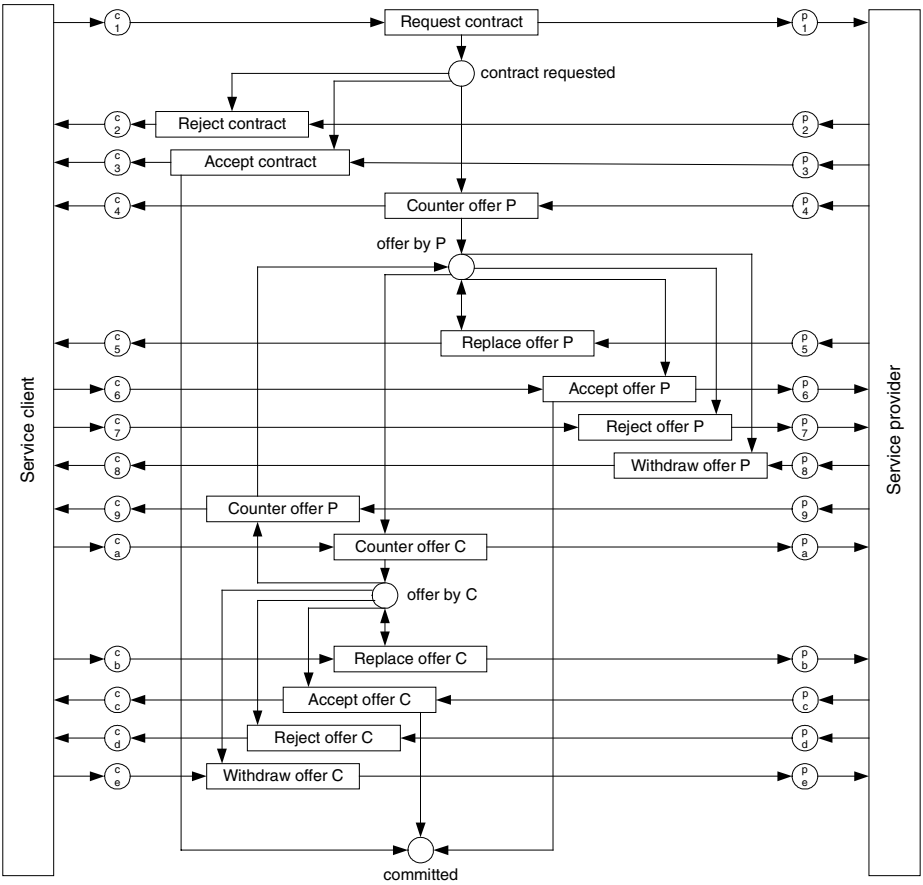

Fig. 13. The 'alternating binding counter offers' protocol pattern

6 Conclusions and Future Research

We have proposed a method for the efficient design of contracting workflows in the configuration process of 'process-aware' ICT components for contracting. The method proposes a standard high-level structure of contracting workflows of which the tasks can be replaced by sub-nets from a library. A prerequisite for this library of sub-nets is a clear standardization of transaction protocols. We propose to standardize a number of protocol patterns on the basis of which specific transaction protocols can be defined. This paper has given an example of possible protocol patterns for the negotiation phase.

Further research has to be conducted to design a set of protocol patterns that is sufficient for the majority of transaction protocols. Mapping of existing transaction protocols to the proposed protocol patterns and the formal standardization process itself are major activities. Another line of research is to extend the Petri Nets defining a transaction protocol with additional quality of service properties (cost of providing services, response times, failure rates, etc.). A transaction protocol extended with these properties can act as an interface agreement between communicating parties. A next step would be the specification of a repository in which a party can publish the transaction protocols that define the external behavior of his services.

References

1. Aalst, W.M.P. van der And K.M. Van Hee, Workflow Management: Models, Methods and Systems, MIT Press, Cambridge, MA, 2001.
2. Dietz, J.L.G., Introduction to DEMO, Samson Bedrijfsinformatie, 1996.
3. Dijk, A. van, The Contracting Agent – concepts and architecture of a generic software component for electronic business based on outsourcing of work, Ph.D. Thesis, Eindhoven University of Technology, 2001.
4. Goldkuhl, G., Generic Business Frameworks and Action Modeling, Proc. of 1st International workshop on Communication Modeling, Springer-Verlag, 1996.
5. Lee, R.M., A Logic Model for Electronic Contracting, Decision Support Systems, Vol. 4, No. 1, pages 27–44, 1988.
6. Merz, M., F. Griffel, T. Tu, S. Müller-Wilken, H. Weinreich, M. Boger and W. Lamersdorf, Supporting Electronic Commerce Transactions with Contracting Ser-vices, International Journal of Cooperative Information Systems, Vol. 7, No. 4, World Scientific, 1998.
7. Normann, R. and R. Ramirez, Designing interactive strategy: from value chain to value constellation, John Wiley & Sons Ltd, 1994.

Security in Business Process Engineering

Michael Backes, Birgit Pfitzmann, and Michael Waidner

IBM Zurich Research Laboratory, Rüschlikon, Switzerland
{mbc,bpf,wmi}@zurich.ibm.com

Abstract. We present a general methodology for integrating arbitrary security requirements in the development of business processes in a both elegant and rigorous way. We show how trust relationships between different parties and their respective security goals can be reflected in a specification, which results in a realistic modeling of business processes in the presence of malicious adversaries. Special attention is given to the incorporation of cryptography in the development process with the main goal of achieving specifications that are sufficiently simple to be suited for formal verification, yet allow for a provably secure cryptographic implementation.

Keywords: Security in Business Process Modeling, Design, Verification and Validation

1 Introduction

For typical distributed business processes, especially those that run over the Internet, from supply chains over business federations to virtual enterprises, security plays a crucial role. For instance, analysts often see lack of security as a major impediment to the adoption of web services. However, the notion of security is often neglected in business-process models, which usually concentrate on modeling the process in a way that functional correctness can be shown, either manually or using formal proof tools like model checking. In contrast to features that are crucial for functional correctness, security features are typically integrated in an application in an ad-hoc manner, often during the actual implementation process. However, this approach brings about several problems.

First, the integration of security features into a development process is not well understood. In particular, a crucial ingredient for adequately modeling security issues – the incorporation of trust assumptions into a specification – is usually not considered, and hence needs further investigation.

Secondly, integrating security properties by hand is difficult and error-prone, and thus lack of experience of individual developers often leads to security leaks. As these developers usually do not have a strong background in security they need to be provided with concrete guidelines and suitable tools for the development of secure applications.

Thirdly, achieving security goals often relies on an appropriate use of cryptographic protocols, e.g., for achieving secrecy for a particular message, the underlying key exchange, or larger cryptographic protocols like payment systems or fair exchange. However, incorporating cryptography already in the specification is surely not desired since

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 168–183, 2003.

this would significantly complicate the use of formal methods in case a validation of security properties is desired. More precisely, a suitable tool would have to cope with probabilism, computational restrictions, error probabilities, and other essential details for reasoning about real cryptographic primitives in a meaningful way. No such tool exists yet. Hence abstractions for such protocols should be provided that are as simple as possible on the one hand, yet extensive enough to allow for being securely implemented on the other hand. One of the main problems in all prior work is the use of oversimplified abstractions, which are insufficient in the sense that they cannot be securely implemented, even if provably secure cryptographic primitives are used. Today, a large variety of applications presupposes the use of cryptographic protocols (but without actually specifying them), which exemplifies the demand for a common formal model that can deal with these issues on the one hand, but can still be conveniently used for expressing and analyzing a large variety of applications.

1.1 Our Contribution

This work presents a guideline how to integrate *arbitrary* security requirements in the specification of business process modeling in a both elegant and rigorous way. Before an application is specified, the developer should be aware of the security goals he wants to achieve. This is shown on the right-hand side of Figure 1. The security properties considered in this work are especially not restricted to commonly analyzed, message-specific properties like secrecy or authenticity of specific message, but comprise sophisticated properties like probabilistic non-interference (absence of probabilistic flow of information), fair exchange, or privacy guarantees based on privacy policies. Although these properties are essential for lots of business processes, their incorporation in the design process has not been well understood yet. Furthermore, the trust relationship between different parties has to be reflected in the design. We investigate the notion of trust models for this purpose, and we show how they can be used to allow for a convenient incorporation of trust into the development process. As shown in Figure 1, after reflecting these features in the specification, common approaches on step-wise refinement can be applied.

Our further work is based on a probabilistic model of reactive networks for expressing and analyzing cryptographic protocols from [26]. By exploiting the benefits of this approach, we show how business processes including security issues can be specified in a way that is suited for formal verification as the use of probabilism that arises from the underlying cryptography can be avoided on this layer without destroying the link to the (necessarily probabilistic) cryptographic implementation. This means that the crypto-related parts of a specification can be refined automatically without any further assistance of the developer yielding a concrete implementation of the system (provided that the parts that are not related to security are sufficiently specified) such that this refinement preserves all security properties of the specification. This is also shown on the right-hand side of Figure 1. Here, a concrete implementation with still abstract representations of cryptography can be easily and securely refined, since the model offers an appropriate interface for a cryptographic library, and the abstract specification provided by the model ensures a correct deployment. As the final result of our whole approach, we obtain a concrete proposal for a secure implementation of the specification including an appropriate deployment of actual cryptography.

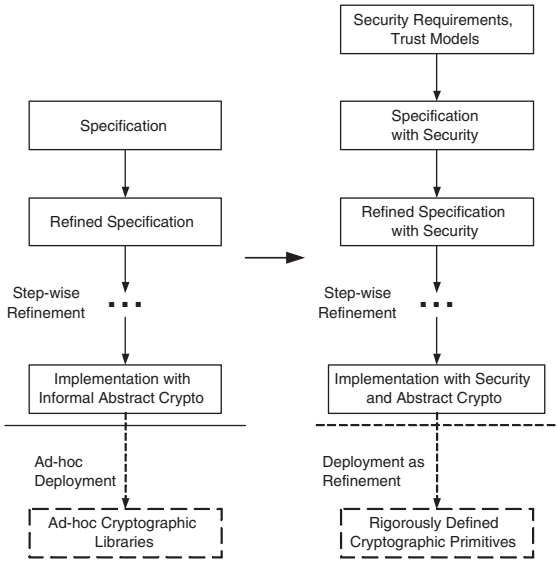

Fig. 1. A usual development approach based on step-wise refinement is shown on the left side. The right side shows the main extensions of our approach, i.e., including security requirements and trust models into specifications, and appropriate refinement of cryptographic abstractions.

This stands in blatant contrast to prior approaches that make implicit or explicit use of real cryptography, in which the development of an application stops to be elaborate if it comes to the implementation of real crypto.

1.2 Prior Work

The early days of business-process modeling were stamped by the use of pictures that described the interaction between different components of the system. This is still very common at a high level of abstraction, but nowadays merging different views on the process within suitable specification languages, usually workflow management systems or UML. However, if formal validation is desired, there is a need to equip such languages with a non-ambiguous semantics before, e.g., model checking techniques can be applied. Hence, research on business process modeling has recently started to encode business-process diagrams into a formal model that can be given a suitable semantics, usually based on interacting state machines [17,11], petri nets [15], or graph grammars [16]. Based on the language ConGolog, a more action-oriented representation of business processes is presented in [18]. However, these works did not explicitly take security aspects into account.

Besides the mentioned models, well-known standards for business modeling like ebXML and WSFL emerged that included security issues by postulating the use of secure and reliable message transmission, but they do not define this notion beyond the informal description, nor do they address more complex security goals. For a nice overview, see [22].

Up to now, models that explicitly address the incorporation of security issues in the design process are typically extensions of a fragment of UML that can be given

the desired semantics. They address more general notions of security than in the standards like multi-level security [14] or role-based access control [19]. Although these are already significant results in this area, several important features like probabilistic flow of information and faithful representation of cryptography as described above have not been addressed yet. However, as some of the above work is at least implicitly based on cryptographic techniques like ensuring authenticity in access control or secrecy for transmitting confidential information in multi-level security, abstraction versions of cryptographic primitives like secure channels are included there, following the approach of Dolev and Yao [10]. Using these abstractions allows for fairly simple proofs of security properties. From the view of cryptography however, these abstractions are unfaithful in the sense that they cannot be securely implemented even if provably secure cryptographic primitives are used. A concrete counterexample is given in [25]. Being more precise, there is no guarantee that properties of these abstractions are also valid for the concrete implementation.

2 Security Requirements and Trust Models

This section investigates the incorporation of security requirements and trust models into the specification of a business process. We further introduce a certifies mail protocol as a running example, which we will consecutively develop as the paper proceeds.

To incorporate security in the development of a business process, several aspects naturally have to be taken into account. First the addressed security properties have to be identified. Up to now, this task usually has been restricted to relatively simple security properties like secrecy of a specific message, but typical processes that are based on more sophisticated protocols often presuppose achieving more general security properties like fair exchange or absence of information flow. Secondly, the relationship of trust between the different parties has to be investigated and incorporated in the design, i.e., which users trust which other users for which purpose. Further comprised by this point is that a developer might be interested in the number of malicious participants that the considered application can successfully tolerate. Dealing with questions of this kind leads to the notion of *trust models*, which are common in the security community, but have not been properly addressed for business process modeling so far.

2.1 Security Goals

Before starting to develop a secure application, the desired security goals have to be determined. Typically, the addressed security goals are restricted to either single-sided security, e.g., by establishing security using a trusted computing base that can be evaluated according to Common Criteria, or to channel- or message-specific properties like secrecy of a particular channel, which can (hopefully) be achieved in a single step by a suitable deployment of encryption schemes. However, this is not sufficient for living up to the multilateral security demands of multiple sides that are running a complex protocol. A typical example in business modeling are fair exchange systems, where different "basic" security goals collide from different sides and have to be achieved at once, e.g., a buyer wants to receive the ordered goods if he properly pays for them whereas a seller wants to receive his money if he delivers the goods. Dealing with such general goals for multiple parties is much more difficult for the developer since it is not sufficient to, e.g.,

simply classify a channel as being secure, but the security property may depend on the continuous interaction of several parties, or may even change over time. Hence, incorporating such properties in formal specification gives rise to several questions, which are often circumvented by including the property in prose without defining it beyond the reach of an informal description. However, this mainly passes on the problem to the implementor who now has to derive a secure implementation without sufficient guidelines by the specification. Hence a formal model for business processes should allow for expressing *arbitrary* security goals, and it should further provide concrete guidelines how a secure implementation can be achieved from a secure specification.

2.2 Trust Models

Typically, a trust model refers to one specific property only, i.e., one system might use different trust models for different properties. For example, a user might trust another user of his own company to proof-read some unpublished work without using it for its own advantage, but not to use his company credit card. Many companies and governments even officially do not have a centralized trust model, i.e., different departments are often not allowed to share all their knowledge and delegate all their tasks to each other.

For a meaningful incorporation of trust in the development process, a trust model has to meet the demands of two different points of view.

First, it has to specify which parties trust each other for which purpose, respectively which parties are considered as being potentially dishonest. This represents the individual demands and relationships between the different users. Obviously, these trust relationships heavily depend on the addressed security requirements, so it should be avoided to lump together all kinds of security requirements, but each requirement may need an individual treatment. This is an important feature in our underlying model, where different requirements can be expressed as integrity, privacy, and liveness requirements. For dealing with security issues in the presence of malicious adversaries, this view usually captures the number of dishonest parties that a system can successfully tolerate without losing its promised security properties. The common way to capture this formally is to use *access structures*. An access structure $\mathcal{ACC}$ is a subset of the powerset of all participants, representing the honest participants or correct hardware components; for the set $\mathcal{M} = \{1, \ldots, n\}$ of participants, e.g., we have $\mathcal{ACC} \subseteq 2^{\mathcal{M}}$. $\mathcal{ACC}$ has to be closed under element insertion, i.e., with $A \in \mathcal{ACC}$ and $A \subseteq B$, we have $B \in \mathcal{ACC}$. For cryptographic protocols, typical examples are threshold structures, which state that at least t parties have to be honest to guarantee the desired property, i.e., $\mathcal{ACC} = \{A \in 2^{\mathcal{M}} \mid |A| \geq t\}$. However, such threshold structures do not always adequately model trust for business processes, since the amount of tolerable malicious parties does not only rely on the actual number of such participants, but it might additionally reflect the users' privileges, rights, and influence, and hence their respective power to attack the system. Obviously, it will usually be much more difficult to keep data secret from a corrupted CEO with all her privileges than from a usual employee. Similarly, corrupting a pivotal server storing sensitive data like secret keys for lots of employees will surely be more attractive than corrupting a stand-alone workstation of a single employee.

Secondly, it has to be specified which information may leak from a communication and how the adversary (respectively the dishonest parties) can interfere in the communication process, e.g., by modifying messages in transit. This corresponds to an external

view of trust, which is captured using a *channel model*. Formally, a channel model is a function χ mapping each connection of a system model to an element of the set $\{\mathsf{s}, \mathsf{r}, \mathsf{i}\}$, representing (s)ecure, (r)eliable, or (i)nsecure channels. The set can be extended to other connection types. Both access structures and channel models will be further treated when we introduce our formal model.

2.3 A Certified Mail System as Running Example

In order to illustrate the need for the previously mentioned security goals, we continue with a simple example system: a certified mail system, which models a fair exchange of a message against a receipt, i.e., the message is delivered if and only if a valid receipt is issued. If this condition is violated, then the cheated person should be able to successfully complain at a trusted third party. Certified mail is an important primitive for electronic-commerce processes and other atomicity services.

Here and in the following, we let $\mathcal{M} := \{1, \ldots, n\}$ denote the set of the users of the certified mail system, and these users can communicate with each other over an insecure network like the Internet. In our certified mail system, this set has to comprise one distinguished trust third party v that has to be honest, i.e., it is trusted by both the sender and the recipient.

In order to implement such a system, each participant at first has to be able to check the authenticity of an incoming message. This obviously cannot be achieved that easily in the presence of malicious adversaries and an insecure network, since it allows for modifying a message and faking its origin without granting the recipient a possibility to detect this tampering. For successfully complaining at the third party in case of cheating, a party has to be forced to issue a commitment before it will actually receive the desired goods. The cheated user can use this commitment to convince the verifier in case of a cheating contractual partner. Cryptographic protocols are well-suited for these tasks.

3 General Model Description

This section contains a brief review of the asynchronous model of probabilistic reactive systems from [26], on which our subsequent work is based. The model itself is very rigorously defined, but we use an informal description here due to lack of space. The model is based on interacting state-machines as a common approach in business process modeling; actually, the whole model has many similarities with other commonly used models. The main difference is the incorporation of probabilism for dealing with the concrete versions of cryptography in a meaningful way.

The machine model is probabilistic state-transition machines, similar to probabilistic I/O automata as sketched by Lynch [21]. In prior formal models for business process modeling, the usage of probabilism is avoided by assuming either deterministic or non-deterministic machines. Although this is reasonable if the model does not intend to address security properties, which are either explicitly or – more often – implicitly based on cryptographic techniques, it does not allow for a meaningful analysis in the presence of cryptography.

Communication between different machines is done via *ports*. Inspired by the CSP-Notation [12], we write output and input ports as $\mathsf{p}!$ and $\mathsf{p}?$ respectively. *Connections* are defined implicitly by naming convention, that is port $\mathsf{p}!$ sends messages to $\mathsf{p}?$. To

achieve asynchronous timing, a message is not directly sent to its recipient, but it is first stored in a special machine $\widetilde{\mathsf{p}}$ called a *buffer* and waits to be scheduled.

If a machine wants to schedule the i-th message of buffer $\widetilde{\mathsf{p}}$ (this machine must have the unique clock out-port $\mathsf{p}^{\triangleleft}!$) it simply sends i at $\mathsf{p}^{\triangleleft}!$. The buffer then schedules the i-th message and removes it from its internal list. In our case, most buffers are scheduled by the adversary, i.e., he has the clock out-port. The concept of buffers is essentially the same as in other state-based business-process models, which simply postulate the network to be asynchronous without mentioning where messages in transit are stored, but reasoning about cryptography in a meaningful way presupposes a higher level of rigorousness. Moreover, note that these buffers do not have to be explicitly taken care of when modeling a system. They are implicitly added in the model to allow for an asynchronous definition of time and a meaningful analysis of security properties.

If a machine is switched, it receives an input tuple at its input ports and performs its transition function yielding a new state and an output tuple in the deterministic case, or a finite distribution over the set of states and possible outputs in the probabilistic case. At each switching step of one particular machine, at most one value can arrive at every input port and the machine can at most produce one output per port.

A *collection* $\mathcal{C}$ of machines is a finite set of machines with pairwise different machine names and disjoint sets of ports. A port of a collection is called *free* if its connecting port is not in the collection. These port will be connected to the users and the adversary. A collection $\mathcal{C}$ is called *closed* if it has no free ports except a special master-clock in-port $\mathsf{clk}^{\triangleleft}?$. This port will be used to resolve situation where the execution cannot proceed.

For a closed collection, runs (sometimes called *traces* or *executions*) are defined as follows. Scheduling of machines is done sequentially, so we have exactly one active machine M at any time. If this machine has cloc out-ports, it is allowed to select the next message to be scheduled as explained above. If that message exists, it is delivered by the buffer and the unique receiving machine is the next active machine. If M tries to schedule multiple messages, only one is taken, and if it schedules none or the message does not exist, the special master scheduler is scheduled using the master-clock in-port $\mathsf{clk}^{\triangleleft}?$. For a formal definition of runs, see [26]. Due to the probabilistic transition functions of the individual machines, we further obtain a probability space over the runs, which will be useful for adequately expressing sophisticated security goals like probabilistic flow of information. We further define the restriction of a run to a set $\hat{M}$ of machines by restricting the run to those steps where a machine $\mathsf{M} \in \hat{M}$ is switched. This is called the *view* of $\hat{M}$.

3.1 Security-Specific System Parts Based on Trust Models

For security purposes, special collections are needed, because an adversary may have taken over parts of the initially intended system. This is handled by considering a *system* to be a set of possible *structures*, i.e., we have one structure for each element of the trust model. First, the actual system part is defined and then the environment, consisting of *users* and the *adversary*. Each structure furthermore separates the set of free ports into *specified ports* S and others. The specified ports are those where a certain service is guaranteed. Typical examples of inputs at specified ports are "send message m to id" for a message transmission system or "pay amount x to id" for a payment system. In

the upcoming security definition, only the events at the specified ports have to be taken care of. This allows *abstract specifications* with *tolerable imperfections*, which can be explicitly granted to the adversary at the remaining ports of the system. A structure is completed to a *configuration* by adding machines H and A, modeling the joint honest users and the adversary, respectively. The machine H is restricted to the specified ports S, A connects to the remaining free ports of the structure and both machines can interact, e.g., in order to model active attacks.

Configurations are always closed, i.e., no inputs or outputs are related to some external participant. This is suitable since users and the adversary are explicitly contained in the configuration, so there is no need to model them externally as often done in other formal models. We will see that including the users in the model allows for using the concept of *simulatability* that, roughly speaking, reflects the notion of a cryptographically secure implementation.

3.2 Standard Cryptographic Systems

In a *standard cryptographic system*, the structures are derived from one *intended structure* and a *trust model*. The intended structure typically consists of n machines M_u, one for each possible user. This is shown on the left-hand side of Figure 2. The trust model consists of the already mentioned access structure $\mathcal{ACC}$ and channel model χ, cf. Section 2.2. Here, the access structure contains the possible sets of uncorrupted machines (among the intended ones), and the channel model classifies each channel as secure, reliable (authentic but not private) or insecure. Now, we have one structure for each element $\mathcal{H}$ of $\mathcal{ACC}$, i.e.,

$$Sys = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$$

in which only the machines contained in $\mathcal{H}$ are considered, i.e., we have $\hat{M}_{\mathcal{H}} = \{\mathsf{M}_u \mid u \in \mathcal{H}\}$. The remaining machines are absorbed into the adversary. Now we modify the channels between two correct machines according to the channel model in the following way. If a connection c is considered as secure ($\chi(c) = \mathsf{s}$), it remains unchanged, i.e., both parties are directly connected, and the adversary does not learn anything about communications on that channel. If a connection is reliable ($\chi(c) = \mathsf{r}$), every output is additionally sent to the adversary corresponding to listening on a channel without the ability of changing the content. This corresponds to an authenticated channel. Finally, messages sent on insecure connections ($\chi(c) = \mathsf{i}$) have to pass through the adversary, i.e., he can read and modify them in transit. A configuration of such a structure, after taking the trust model into account, is shown on the right-hand side of Figure 2. After these modifications, each derivation represents a realistic scenario for the particular trust model, where information may leak or authenticity may not be guaranteed because of an insecure network like the Internet, or where exchange of public keys can be precisely modeled using authenticated channels.

3.3 Simulatability

Simulatability is an important cryptographic concept, which allows for securely refining the cryptographic parts of a specification. Simulatability essentially means that whatever

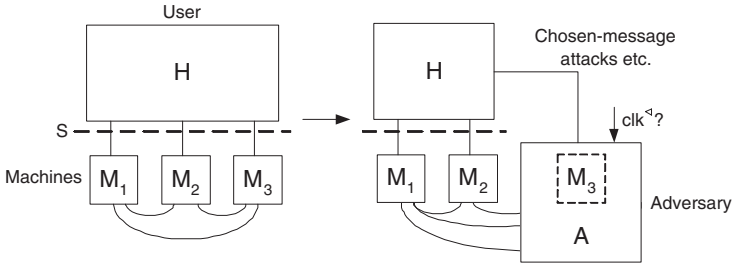

Fig. 2. A configuration of an intended structure is shown on the left-hand side (for $n = 3$). The right-hand side shows a derivation induced by the trust model. In this case, we have $\mathcal{H} = \{1, 2\}$, so the third machine is absorbed into the adversary, and the channel model classifies the connection between the correct machine as authenticated.

might happen to an honest user H in a real system Sys_{real} can also happen to the same honest user in an ideal System Sys_{id}. Formally speaking, for every configuration $conf_1$ of Sys_{real} there is a configuration $conf_2$ of Sys_{id} with the same users yielding *indistinguishable views* of H in both systems. We abbreviate this by $Sys_{\mathsf{real}} \geq_{\mathsf{sec}} Sys_{\mathsf{id}}$. Indistinguishability is a well-defined cryptographic notion from [29].

Several nice results on simulatability exist. At first, there exists a composition theorem stating that a step-wise refinement maintains the simulatability property [26]. Moreover, integrity and privacy properties (more precisely, integrity properties formulated in a linear-time logic and non-interference properties) are preserved under simulatability[4, 2], which allows for proving properties on a high-level abstract layer that can be refined later down to the cryptographic layer without destroying the already proven properties. These theorems are essential for modular proofs.

Hence, the analysis of security properties only has to be done for the abstract specification, and the results magically carry over to the actual cryptographic implementation. In the upcoming section, this approach is used to define abstractions of commonly used cryptographic primitives, which can be implemented in the sense of simulatability. The benefit is that these abstractions can be conveniently used within a specification, and serve as a construction kit for designing large processes out of these small building blocks. Refining the crypto-related parts can then be easily, but securely achieved without any additional work. Hence designers or process analysts are not forced to have a strong background in security.

4 Towards a Secure Implementation: Refinement Using Common Cryptographic Primitives

In this section, we investigate the notion of refinement of a process within our model, distinguishing between refinement on the component level, on the action level, or on the trust level. As the refinement of abstract cryptographic specifications by real cryptographic protocols is one of the main benefits of our model, we continue with a brief review on abstract and concrete versions of the most important cryptographic primitives comprising secure channels, certified mail, and a simulatable cryptographic library.

4.1 Security in Different Layers of Refinement

Refinement is an important aspect of a design process, as shown in Figure 1. The part where our model is most beneficial is that it enables sound refinement of abstract specifications of cryptographic protocols by real cryptographic protocols. However, the inclusion of trust and security requirements may influence all layers.

Another important argument for having a joint model of business processes and cryptographic protocols is that typical business-process modeling may occur again *under* a layer with cryptographic protocols.

All this is illustrated in Figure 3. Part 1 of this figure shows an abstract cryptographic

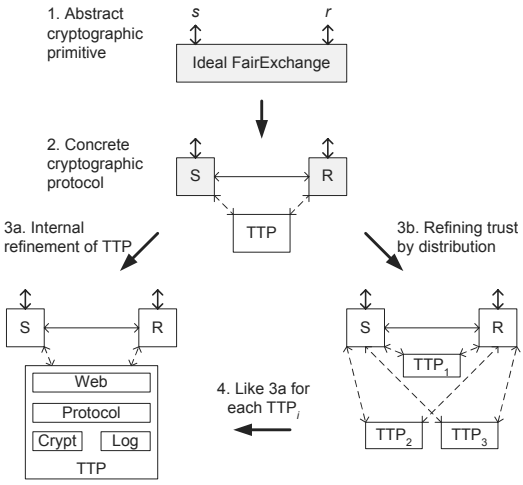

Fig. 3. Different types of refinement.

primitive, here fair exchange as sketched in Section 2.3. In an overall business-modeling process, this may be one element after previous refinements.

Part 2 of the figure shows refinement by a real cryptographic protocol. This is primarily component refinement because the behavior of the abstract primitive is modeled by one state machine, while the real protocol shows the real three parties of such a protocol. Further, it contains action refinement: On the abstract layer, the fair exchange is essentially one step (exchange or not, depending on certain preconditions; only asynchrony and giving the adversary a choice to disrupt the protocol breaks this step up a bit). The real protocol contains several real state transitions corresponding to an input of a protocol message, an internal action of one party, and output of the next protocol message.

We show two possible next refinement steps: Part 3a of the figure shows business-process modeling of the party TTP. While in the cryptographic protocol its action is only a few monolithic state transitions, for a real such party quite complicated internal workflows must be designed. The figure just shows a web-service frontend, a protocol engine, a crypto engine and a logging component that interact to implement the state transitions. Further, one needs management components for these functional components

and approval workflows for use of the management components. This may be treated as a standard business-process modeling and implementation issue without further specific security considerations, because the trust model in Layer 2 was that users s and r have to trust TTP. However, it is better to refine the trust model on this layer, so that nobody needs to trust all components of TTP. For instance, web-service frontends are less trusted than crypto engines. "Refinement" for trust means that no new trust may be introduced. More precisely, the system with the trust model inherited from the higher layer must fulfill the higher-layer properties, but the more we can restrict the necessary trust in sub-components the better.

Part 3b shows another way of refining the trust: Requiring s and r to trust TTP, even with respect to the organization as such and not its components, is quite a strong restriction. We may want to distribute this power. The figure shows refinement of TTP by a cryptographic protocol for secure state-machine distribution with 2-out-of-3 trust, i.e., as long as two of the three organizations TTP_1, TTP_2, and TTP_3 are trustworthy, the overall protocol is as secure as that on the higher layer. This goes along with a refinement of the messaging interface of components S and R to interact with the distributed TTP. (See [8] for real protocols achieving this.) After this step, for each individual TTP_i, business-process modeling as in Step 3a may be applied.

4.2 Secure Channels

In the following, we review the fundamental primitive of secure channels, which will probably serve as the foundation of upcoming modeling examples due to its simplicity but generality. We start with the concrete implementation, and continue with a deterministic, but faithful abstraction, i.e., properties proved for this specification carry over to the concrete implementation. It was a longstanding open question whether such "benign" abstractions exist, which was answered in the affirmative for some of the most important cryptographic primitives so far. The solution to the problem was to augment naive abstractions with tolerable imperfections to obtain idealized systems for which practical protocols exist.

Concrete Implementation. Our real system is a standard cryptographic system of the form where any subset of participants may be dishonest. It uses asymmetric encryption and digital signatures as cryptographic primitives. A user u can let his machine create signature and encryption keys that are sent to other users over authenticated channels. Messages sent from user u to user v are signed and encrypted by M_u and sent to M_v over an insecure channel, representing a real network. The adversary can schedule the communication between correct machines and send arbitrary messages to arbitrary users (but only using the identity of the dishonest users).

Abstract Specification. The abstract specification is a system of the form $Sys_{\mathsf{ideal}}^{\mathsf{SecMess}} = (\mathsf{TH}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$ where $\mathsf{TH}_{\mathcal{H}}$ is a deterministic machine modeling an idealized behavior of the real system. A user can again let $\mathsf{TH}_{\mathcal{H}}$ create signature and encryption keys, or send messages to arbitrary users. In contrast to the real system, no keys are actually generated, but $\mathsf{TH}_{\mathcal{H}}$ only stores that a key generation of the particular user has happened and informs the adversary of this fact. If a message should be sent from user u to user v then this message is not sent over the network, but simply stored in

an internal array of $\mathsf{TH}_{\mathcal{H}}$. Now $\mathsf{TH}_{\mathcal{H}}$ tells the adversary that a message has been sent from u to v, and gives it a handle to the particular position in the array that contains the message, along with the actual length of the plaintext message. The adversary can use this information to schedule the message, which is then delivered by the trusted host.

Intuitively, the machine $\mathsf{TH}_{\mathcal{H}}$ models the key essence of sending of encrypted messages and authenticated key exchange. The adversary does neither learn anything on the content of a sent message nor does he has the possibility to alter it, but he is explicitly granted certain information like the length of the plaintext, and that a message has indeed been sent. These are the already mentioned tolerable imperfections, which stem from the fact that a cryptographic implementation cannot avoid the adversary from learning this particular information on his own, at least not with reasonable loss of complexity. Hence, modeling these imperfections also within the specification is the key element for achieving secure implementations.

Some Variants. For secure channels, several variants have already been proven with a simulatability definition like ordered channels [3], which guarantee maintaining the order of sent messages, or reliable channels [5], which guarantee eventual delivery of a message. One of the most important extension is the cryptographic firewall [4] which uses digital signatures to establish a virtual private network, i.e., it builds up a firewall around users which are only connected via an insecure network like the Internet. The firewall is mainly achieved be augmenting the secure channel primitive with a filtering system that sorts out message, which are sent by senders outside of the firewall. The secure channels already provide authenticity of messages and their origins, hence the filtering procedure precisely sorts out the desired message. However, certain complications have to be taken into account like avoiding denial-of-service attacks etc.

4.3 A Primitive for Certified Mail

Now we review the primitive for our running example of certified mail.

Concrete Implementation. The real system is again a standard cryptographic system; however, now a specific party TTP must be honest, i.e., all sets in $\mathcal{ACC}$ contain it.

The system starts with a key-exchange phase. Later, for each exchange, the sender s and recipient r run a subprotocol. As an example, we sketch the efficient asynchronous protocol from [1].

It consists of a four-message standard flow, and two subprotocols where one of the original parties complains to TTP, and TTP fairly aborts or finalizes this exchange. In Message 1, the sender gives a commitment to the mail. A commitment is a lower cryptographic primitive that fixes a message, but, like encryption, does not allow the message to be read yet. In Message 2, the recipient signs that he is willing to provide a receipt if this commitment is opened to him. In Message 3, the sender opens the commitment, i.e., sends the message and a verification value, and in Message 4, the recipient sends a receipt.

If the recipient does not send the receipt (or due to the asynchronous network or an outside adversary it is delayed for too long), the sender shows Messages 1 to 3 to TTP. If the recipient has sent the promise (Message 2) and does not receive the opened commitment, he also complains to TTP with Messages 1 and 2. If a sender-complaint

arrives first, TTP provides a replacement receipt. If a recipient-complaint arrives first, TTP signs for the recipient that the exchange was aborted.

Abstract Specification. The abstract specification again essentially consists of one deterministic trusted host $\mathsf{TH}_{\mathcal{H}}^{\mathsf{CertMail}}$ in each structure. The main inputs are of the form (send, r, m) for sending a mail m to a recipient r, and $(\mathsf{receive}, s)$ for a recipient to indicate willingness to get a mail from s and provide a receipt. Essentially, the trusted host looks whether there are two matching inputs; if yes, it outputs the message m to r and a success indicator to s. A detailed specification is already subtle and interesting for the higher processes. First, we need transaction identifiers as additional parameters to define which inputs match. Secondly, we have to choose between the simple inputs just defined and a so-called labeled variant where the recipient also agrees to a subject for the mail to be received, i.e., the above inputs have yet additional parameters. This may be of more use in many applications. Thirdly, there are inputs to show and verify a receipt, because that is the purpose of receipts in a higher protocol. Finally, we have to model the power of an adversary in realistic protocols like the one above: He may learn of exchanges even between honest participants, but not the message and subject involved, and he can force protocols to end unsuccessfully.

4.4 A Foundation for Sophisticated Cryptographic Protocols: A Cryptographic Library

When using more sophisticated protocols, we cannot only rely on simple secure channels, since several messages might be encrypted twofold, nonces might have to be included etc. Moreover, plugging in cryptographic primitives in larger protocols in a naive way may give rise to man-in-the-middle attacks, type confusion attacks, and so on. Hence, we need a more sophisticated system that takes care of all these subtleties, mainly by following the rules of robust protocol design. Moreover, it should allow for composing messages, for including important design principles like nonces etc.

Recently, a simulatable cryptographic library has been introduced in [6], which provides these important design principles, and serves as a powerful tool for integrating security in the analysis (and the design) of business processes. We omit a more detailed description due to lack of space.

4.5 Modeling the Certified Mail System

In the following, we express the certified mail system of Section 2.3 in our model. The system offers each user one port for sending and receiving messages from the system, respectively. Using the conventions of [26], these ports are named $\mathsf{out}_u!$ and $\mathsf{in}_u?$ for user u. (The corresponding ports $\mathsf{out}_u?$ and $\mathsf{in}_u!$ are then ports of the system.) These ports are also the specified ports of the system. Using the abstract primitive $\mathsf{TH}^{\mathsf{CertMail}}$ for certified mail from Section 4.3, we can now easily derive an abstract version of our system that has the desired functionality. This is shown at the left-hand side of Figure 4. The derivation of a concrete implementation is also very simple. We just have to replace the abstract system $\mathsf{TH}^{\mathsf{CertMail}}$ with its predefined secure implementation, consisting of the actual protocol machines $\mathsf{M}_u^{\mathsf{CertMail}}$. Since this implementation has been derived according to the simulatability paradigm (cf. Section 3.3), the desired integrity property

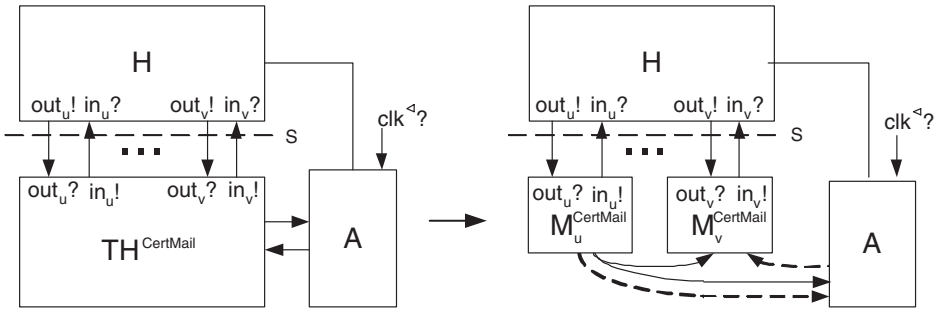

Fig. 4. Refinement of the Certified Mail Specification. The abstract primitive is replaced by its concrete cryptographic counterpart.

of certified mail carries over to the implementation without any further work, i.e., this implementation is provably secure if the specification can be proven to be secure. Hence, the final step is the actual validation of the specification, preferably using formal methods. The feasibility of formal methods for this task is the topic of the next section.

5 On the Applicability of Formal Methods

The formal verification of security issues for arbitrary (non business-related) processes has been subject to lots of papers in the literature (a very partial list includes [27, 20,24]). Just as we did in the previous section, the use of actual cryptography was avoided by considering abstractions. However, the used abstractions are oversimplified in the sense that no secure implementation of them is known. Thus the above concepts for formal verification cannot be easily transfered to the demands of business process modeling, which was one of the main reasons why we decided to base our work on a more cryptographic approach. The disadvantage is that applying formal methods to the abstractions of the previous section is more time-consuming, and also not investigated that well.

As already mentioned in the previous section, all our abstractions are at least deterministic hence we do not have to consider tools for dealing with probabilism like probabilistic model checking. Moreover, no such tool exists for dealing with the cryptographic details like error probabilities. Because of avoiding these problems, our abstractions are in scope of formal proof systems, at least of formal theorem provers, for which significant results on proving medium-size examples of this kind already exist, e.g., in [3,2] using the theorem prover PVS [23]. All of our abstractions presented above are surely in scope of theorem proving, and further work in this area may additionally benefit from specialized tools such as SAL [7] – an extension of PVS – that provides an environment for the analysis of systems specified as state machines.

However, the use of automated model checking instead of theorem proving would surely be very promising as it has become a very popular method to verify the properties of finite-state concurrent systems [9]. It is also the more common approach in the business process community, mainly because it has been successfully used in other models to verify medium-sized examples [13,17]. Since interacting finite-state machines are the foundation of these techniques, they are as well applicable to our underlying model. More

precisely, if we avoid using our cryptographic abstractions from the previous section, and concentrate on modeling common examples without security just as prior work does, using our model for this task does not impose any disadvantages.

When using our abstractions, the main problem could be that they are simply too complex for being model-checked, which hence needs further investigation. However, the basic primitives like secure channels or the cryptographic firewall are surely feasible for current model checkers. In contrast to that, the more sophisticated abstractions like certified mail or even the cryptographic library have very complex transition functions that are additionally based on complex and unbounded datastructure like a database with different types of entries. An ad-hoc application of a model checker will fail almost surely, but we are confident that the complexity can be further reduced by developing or adapting commonly data-independence techniques, e.g., [28].

References

1. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proc. 19th IEEE Symposium on Security & Privacy*, pages 86–99, 1998.
2. M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686. Springer, 2003.
3. M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In *Proc. 11th Symposium on Formal Methods Europe (FME 2002)*, volume 2391 of *Lecture Notes in Computer Science*, pages 310–329. Springer, 2002.
4. M. Backes and B. Pfitzmann. Computational probabilistic non-interference. In *Proc. 7th European Symposium on Research in Computer Security (ESORICS)*, volume 2502 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2002.
5. M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial fairness and liveness. In *Proc. 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 160–174, 2002.
6. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. IACR Cryptology ePrint Archive 2003/015, Jan. 2003. `http://eprint.iacr.org/`.
7. S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muñoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari. An overview of SAL. In *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, 2000.
8. C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the Internet. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, pages 167–176, 2002.
9. E. Clark, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.
10. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
11. D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The Statemate Approach*. McGraw-Hill, 1998.
12. C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science, Prentice Hall, Hemel Hempstead, 1985.
13. W. Janssen, R. Mateescu, S. Mauw, P. Fennema, and P. van der Stappen. Model checking for managers. In *Proc. Theoretical and Practical Aspects of SPIN Model Checking*, volume 1680 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 1999.
14. J. Jürjens. Towards development of secure systems using UMLsec. In *Proc. Fundamental Approaches for Software Engineering (FASE)*, pages 187–200, 2001.

15. E. Kindler and T. Vesper. A temporal logic for events and states. In *Proc. 19th International Conference on Application and Theory of Petri Nets*, volume 1420 of *Lecture Notes in Computer Science*, pages 365–384. Springer, 1998.
16. C. Klauck and H.-J. Mueller. Formal business process engineering based on grammer graphs. *International Journal on Production Economics*, 50:129–140, 1997.
17. J. Koehler, G. Tirenni, and S. Kumaran. From business process model to consistent implementation: A case for formal verification methods. In *Proc. 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 96–106, 2002.
18. M. Koubarakis and D. Plexousakis. A formal model for business process modelling and design. In *Proc. Conference on Advanced Information System Engineering*, pages 142–156, 2000.
19. T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *Proc. 5th International Conference on the Unified Modeling Language*, volume 2460 of *Lecture Notes in Computer Science*, pages 425–441. Springer, 2002.
20. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.
21. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Francisco, 1996.
22. D. O'Riordan. Business process standards for web services. available at http://www.webservicesarchitect.com/content/articles/BPSFWSBDO.pdf.
23. S. Owre, N. Shankar, and J. M. Rushby. PVS: A prototype verification system. In *Proc. 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. springer, 1992.
24. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
25. B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic security of reactive systems. Presented at the DERA/RHUL Workshop on Secure Architectures and Information Flow, Electronic Notes in Theoretical Computer Science (ENTCS), March 2000. `http://www.elsevier.nl/cas/tree/store/tcs/free/noncas/pc/menu.htm`.
26. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001.
27. A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop (CSFW)*, pages 98–107, 1995.
28. A. W. Roscoe and P. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7(2,3):147–190, 1998.
29. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.

Query Nets: Interacting Workflow Modules That Ensure Global Termination

Rob J. van Glabbeek and David G. Stork

Ricoh Innovations
2882 Sand Hill Rd. Suite 115
Menlo Park, CA 94025-7022, USA
rvg@cs.stanford.edu
stork@rii.ricoh.com

Abstract. We address cross-organizational workflows, such as document workflows, which consist of multiple workflow modules each of which can interact with others by sending and receiving messages. Our goal is to guarantee that the global workflow network has properties such as termination while merely requiring properties that can be checked locally in individual modules. The resulting *query nets* are based on predicate/transition Petri nets and implement formal constructs for business rules, thereby ensuring such global termination. Our method does not require the notion of a global specification, as employed by Kindler, Martens and Reisig.

Introduction

This paper deals with the formal modeling of business processes that transform an input into an output by performing several tasks, or by delegating these tasks to other such business processes. Examples are an insurance company, that takes as input a claim and yields as output a decision on that claim, or a car dealership that takes as input a broken car and yields as output a repaired one. Both businesses follow a pre-established protocol to transform input into output. Such a protocol is called a *workflow*.

In this paper we focus on workflows that span several organizations. Each organization employs a local workflow, and these local workflows may delegate tasks to local workflows of other organizations. This is done by presenting an input to the other organization's workflow, awaiting the generation of an output, and proceeding with the task at hand using that output. (This paradigm is essentially the *Nested Subprocesses Model*, the second interoperability scenario identified by the Workflow Management Coalition [8].) Naturally, our work also applies to modular workflows within one organization.

Take as example the car dealership. The car dealership's workflow is initiated by a customer dropping off a broken car. In case the car dealership can repair the car, that's what they do; otherwise the problem is described to the car's manufacturer in Detroit, and their answer will be used to carry out the repair. The repaired car constitutes the output of the workflow; it can be picked up by the customer. The manufacturer's workflow can be initiated by a question about the repair of a car, and its output is an answer to that question. Some questions are dealt with by asking the local dealership that is supposed to carry out the repair, and using the answer of the dealership as output.

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 184–199, 2003.

In this paper we allow workflows that can deal with different types of input and output. The dealership's workflow for instance can also accept a client shopping for a car as input, in which case the output may be a sale. Or it can take anyone's question on how to repair a car as input, and present an answer to that question as output. In the latter case, difficult questions may be dealt with by asking the manufacturer.

The described workflows of the car dealership and the manufacturer are graphically displayed in Fig. 1. In Section 1 we will formalize this representation as a Petri net.

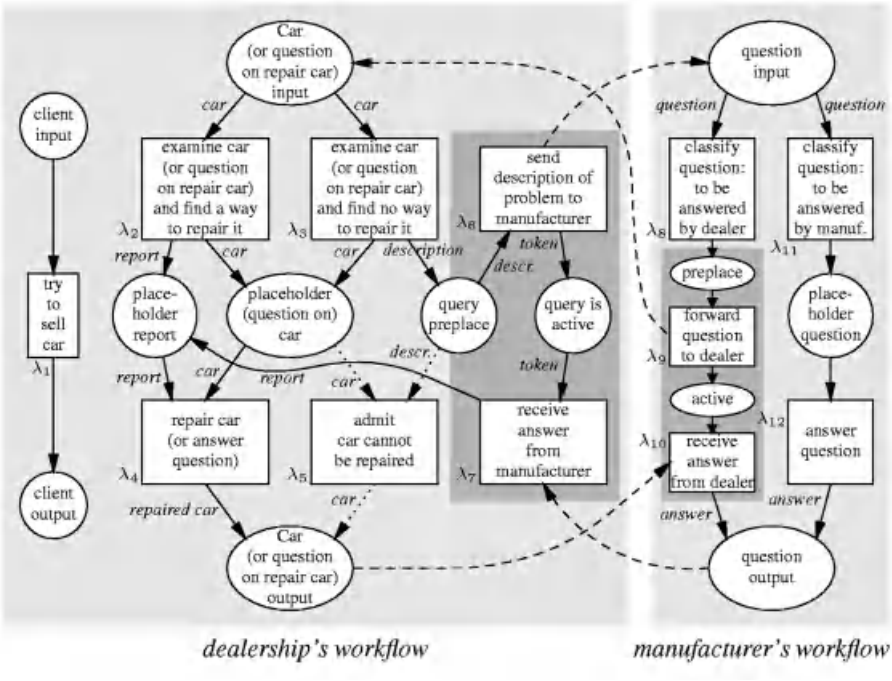

Fig. 1. *Petri net representation of the car dealership's workflow and the manufacturer's workflow, as well as their interaction (the dashed arcs). The dealership accepts as input a client shopping for a car, a broken car, or a question on the repair of a car. As the latter two types of input are treated similarly, they are collected in the same input place.*
Faced with an input of a car, the first task of the dealership is to examine it. Depending on whether the exam yields a way to repair the car or not, the output of this task is either the car and a report on how to fix the car, or the car and a description of the problem for which no fix has been found. In Petri nets, tasks are represented by transitions (the boxes) whose output types are fixed, so the examination needs to be modeled by two transitions.
The shaded part of the dealership's workflow deals with the querying of the manufacturer. With dotted lines we added the alternative of returning the car to the customer unrepaired.

An important property of a workflow is *termination*. On any given input the workflow should eventually produce an output and stop. Let's continue the example of the dealership. If a repair does not succeed, the workflow may prescribe to try to repair the

car again, possibly using another method. When faced with a car that can never be fixed, such a workflow leads to an unending series of attempts to repair the car, and the process will not terminate. This is unacceptable behavior. When all reasonable methods have been tried, the dealership should give up and return the car to the customer unrepaired. Workflows should embody protocols to avoid infinite loops and ensure termination.

When the management of the car dealership inspects its workflow to see if termination is ensured, it has little information on the workflow used by the manufacturer, which is called as a subroutine. Yet, if the manufacturer's workflow fails to terminate, so will the dealership's workflow. Given the workflow sketched above, and the absence of time-outs or other tricks to ensure termination of the invocation of the manufacturer's workflow, the best that can be ensured is a conditional termination property. The car dealership's workflow may be shown to terminate on the condition that the manufacturer's workflow does. We call this *local termination*.

It is in the car dealership's interest to ensure *global* or unconditional termination of its workflow. This can be achieved by verifying that the car dealership's workflow is locally terminating, and the manufacturer's workflow is unconditionally terminating. When the car dealership trusts the manufacturer enough, the verification of the termination of the manufacturer's workflow may happen locally by the manufacturer, and the verdict taken on faith by the car dealership. However, verifying global termination of the manufacturer's workflow may not be so easy. It could be that the manufacturer delegates tasks to others, and those might delegate subtasks even further. In general we are dealing with a network of local workflows, and it may be verified locally that each local workflow in the network is locally terminating, i.e., terminating on the condition that all other workflows in the network are terminating. The question now arises whether such local termination properties of the local workflows are sufficient to guarantee termination of the global workflow.

This needs not always be the case. In could for instance be that the dealership receives as input a car that they do not know how to repair. According to protocol, they ask the manufacturer how to deal with this specific problem. The manufacturer on the other hand may have classified this question as one that ought to be answered by the local dealership. According to protocol, the manufacturer's workflow will pose the question to the local dealership that asked it in the first place. Strictly following protocol, the dealership now deals with the question by asking the manufacturer (again), and an infinite loop results. Thus the car dropped of at the dealership will never be repaired and the global workflow fails to terminate.

In practice, an infinite loop as sketched above will be prevented by some *business rule*. The dealership will never ask the same question twice to the manufacturer in the course of dealing with the same repair. Once the manufacturer echoes back the dealership's original question, the dealership either asks a different question to the manufacturer—a question that doesn't lend itself to being thoughtlessly echoed back to the dealership—or it solves the problem at hand without involving the manufacturer.

The current paper presents a framework for specifying cross-organizational workflows in which such business rules can be implemented. It introduces the concept of a *query net*, which is a locally terminating workflow module that, when embedded in a cross-organizational workflow, will never delegate the same task twice to another workflow module. We establish that in cross-organizational workflow networks built out of such query nets global termination is ensured.

In order to ensure global termination of a cross-organizational workflow network, all that is required is that the individual workflow modules in the network are query nets, a requirement that can be checked locally for each of these modules. Unlike in the work of Kindler, Martens and Reisig [6] there is no need to show correctness of the modules with respect to a *fairness-closed specification*, specifying the interactions between all modules in the global network.

1 Cross-Organizational Workflow Nets

This section presents a Petri net model for cross-organizational workflow. For the purpose of establishing notation and terminology we start out by reviewing basic *place/transition* Petri nets. Then we present a form of the more powerful *predicate/transition nets*, which regard tokens as structured entities. Finally, we arrive at our model by equipping these nets with input and output places and a novel mechanism and implicit protocol for one net to call another as a subroutine.

1.1 Place/Transition Nets

A *place/transition net* is a tuple (S, T, F) where S and T are disjoint sets of *places* (*Stellen* in German) and *transitions*, and $F\colon (S \times T) \cup (T \times S) \to \mathbb{N}$ is the *flow relation*. The elements of S and T are represented graphically by circles and boxes, respectively. For $p, q \in S \cup T$ there are $F(p, q)$ *arcs* from x to y.

When a place/transition net represents a concurrent system, a global state of such a system is given as a *marking* $M\colon S \to \mathbb{N}$. Such a state is depicted by placing $M(s)$ *tokens* in each place s. A *marked place/transition net* is a tuple (S, T, F, M) comprising a place/transition net (S, T, F) and a marking M.

For two markings M and $M'\colon S \to \mathbb{N}$ we write $M \subseteq M'$ if $M(s) \leq M'(s)$ for all $s \in S$. The marking $M + M'\colon S \to \mathbb{N}$ is given by $(M + M')(s) = M(s) + M'(s)$. The function $M - M'\colon S \to \mathbb{Z}$ is given by $(M - M')(s) = M(s) - M'(s)$; this function need not yield a marking because it might specify a negative number of tokens in a place.

The multisets of *preplaces* ${}^\bullet t$ and *postplaces* $t^\bullet\colon S \to \mathbb{N}$ of a transition $t \in T$ in a place/transition net are given by ${}^\bullet t(s) = F(s, t)$ and $t^\bullet(s) = F(t, s)$ for $s \in S$. A transition t is *enabled* under a marking M, written $M[t\rangle$, if ${}^\bullet t \subseteq M$. In that case t can *fire* under M, yielding the marking $M' = M - {}^\bullet t + t^\bullet$, written $M[t\rangle M'$.

If a transition t fires, for every arc from a place s to t, a token moves along that arc from s to t. These tokens are consumed during the firing, but also new tokens are created, namely one for every outgoing arc of t. These new tokens end up in the places at the end of those arcs. The firing of t is possible only if there are sufficiently many tokens in the preplaces of t.

1.2 Predicate/Transition Nets

A *predicate/transition net* [3] (sometimes called *coloured Petri net* [5]) is a Petri net in which the tokens are structured entities. We use a set of *variables* $V = \{x, y, \ldots\}$ to range over the possible tokens in such a net, and a suitable collection $\mathbb{F}$ of formulas over

those variables, expressing combinations of properties of the tokens referenced by the variables occurring in those formulas. The specification of $\mathbb{F}$ varies with the application.

A predicate/transition net is given as a quadruple (S, T, F, λ) where, as above, S and T are disjoint sets of places and transitions, but now the flow relation F is a subset of $(S \times V \times T) \cup (T \times V \times S)$, and λ: $T \to \mathbb{F}$ allocates to each transition a formula called the *transition guard* [7]. As above, the elements of S and T are represented graphically by circles and boxes, respectively, while an element $(p, x, q) \in F$ is represented as an *arc* from p to q, labeled with variable x. A formula $\lambda(t)$ is written next to the box representing the transition t.

An arc $(s, x, t) \in F$ with $s \in S$, $x \in V$ and $t \in T$ indicates that upon firing the transition t, a token x is taken from place s. An arc $(t, y, s') \in F$ with $t \in T$, $y \in V$ and $s' \in S$ indicates that upon firing t a token y is deposited in place s'. The transition guard $\lambda(t)$ selects properties of the input tokens that have to be satisfied for the transition to fire, and simultaneously specifies the relation between the input and the output tokens. The formula $\lambda(t)$ may contain free occurrences of the variables allocated to the arcs leading to or from t. These variables refer to the transition's input and output tokens. Consider as an example a transition that consumes input tokens x and y, and produces an output token z. The transition guard could then be a formula that says that the transition may only fire if y is a cryptographic key, and x is a PGP document that successfully decrypts with that key; if these conditions are met, the decrypted document z is emitted.

The Petri net of Fig. 1 is a predicate/transition net. The arc-labels such as "car" and "report" are variables. The transition guard λ_2 says that the variable "car" refers to a car for which a repair can be found, or alternatively to a question on the repair of a car for which a repair can be found; if this condition is not satisfied, the λ_2-labeled transition cannot fire. Moreover, λ_2 specifies the relation between the car and the report on how to fix it. The same variable "car" labels both an input and an output arc of the transition; this indicates that the car or question passes through this transition unchanged.

1.3 Marked Predicate/Transition Nets and the Firing Rule

A framework for predicate/transition nets specifies a set V of variables, a collection $\mathbb{F}$ of formulas over V, a domain $\mathcal{D}$ of possible tokens, and an *evaluation function*. The latter specifies, for each formula $\varphi \in \mathbb{F}$ and each assignment ξ: $V \to \mathcal{D}$ of tokens to variables, whether φ evaluates to `true` of `false` under ξ; let $\varphi[\xi]$ denote the result of this evaluation. Any specification of a predicate/transition net presupposes such a framework.

A *marking* of a predicate/transition net is an allocation of tokens to the places of the net, formally defined as a function M: $S \times \mathcal{D} \to \mathbb{N}$, that specifies for every place $s \in S$ and token $d \in \mathcal{D}$ how many copies of d reside in s. A *marked* predicate/transition net is a tuple (S,T,F,λ,M) comprising a predicate/transition net (S,T,F,λ) and a marking M.

In a marked predicate/transition net a transition t can fire if there is an assignment ξ of tokens to variables such that the transition guard $\lambda(t)$ evaluates to `true`, and for every x-labeled arc from a place s to t there is an *input* token $\xi(x)$ in s. As a result of firing t, these input tokens are taken away and for every y-labeled arc from t to a place s' an *output* token $\xi(y)$ is deposited in s'.

Here is a more formal description of the firing rule, where the notions $M \subseteq M'$, $M + M'$ and $M - M'$ are defined just as for markings with only one argument. For a

transition $t \in T$ in a predicate/transition net and an assignment $\xi\ V \to \mathcal{D}$, the *input* and *output* markings ${}^\bullet t[\xi]$ and $t[\xi]^\bullet : S \times \mathcal{D} \to \mathbb{N}$ of t under ξ are given by

$${}^\bullet t[\xi] = \{\!\{(s, \xi(x)) \mid (s, x, t) \in F\}\!\} \qquad \text{and} \qquad t[\xi]^\bullet = \{\!\{(s, \xi(x)) \mid (t, x, s) \in F\}\!\}$$

in which $\{\!\{, \}\!\}$ are multiset brackets. A transition t is *enabled* under a marking M, written $M[t\rangle$, if there exists an assignment ξ: $V \to \mathcal{D}$ such that $\lambda(t)[\xi]$ is `true` and ${}^\bullet t[\xi] \subseteq M$. In that case t can *fire* under M, yielding the marking $M' = M - {}^\bullet t[\xi] + t[\xi]^\bullet$, written $M[t\rangle M'$.

1.4 Fairness

A *firing sequence* in a Petri net is a possibly infinite alternating sequence of markings and transitions $M_0, t_1, M_1, t_2, M_2, \ldots$ such that $M_0[t_1\rangle M_1[t_2\rangle M_2 \cdots$. We say that marking M' is *reachable* from marking M if there is a firing sequence starting with M and ending with M'.

In a Petri net, complete runs of the represented system are represented by firing sequences. However, not every firing sequence represents a complete run. Some finite firing sequences merely represent partial runs, and some infinite ones do not correspond with runs that could occur in practice. Those firing sequences that do model complete runs are called *fair*. Fairness can be formalized in many ways, often requiring a more involved definition of a Petri net, and often depending on certain *progress* or *fairness* assumptions made on the behavior of nets. In our simple nets we call a firing sequence *fair* if there is no transition t such that from a certain marking in the sequence onwards, t is continuously enabled but never fires. In particular, a finite firing sequence is fair if and only if in its last marking no transition is enabled. For a more subtle approach to fairness see for example Kindler, Martens & Reisig [6].

1.5 Workflow Nets

Workflow nets are defined in van der Aalst [1,2] employing place/transition nets. Here we extend the definition to predicate/transition nets. A *workflow net* (S, T, F, λ, i, o) is a predicate/transition net (S, T, F, λ) with two special places, i and o. A workflow net represents a business process that converts an input into an output. The input and output are represented by tokens, presented by the environment to the net's *input place* i, and removed from its *output place* o. The specification of a workflow nets does not involve the notion of an *initial marking*; a workflow net starts out empty, and may start firing using tokens dropped in the input place i by the environment of the net.

1.6 Multi-organizational Workflow Nets

In this paper we view the parallel composition of any number of business processes as a single global business process. Imagine two shops, each with their own procedures for handing input and output. While we generally consider them as two separate business processes, we might choose instead to treat them as a single global business process, perhaps because of joint ownership or some other binding relation between the two. When representing processes as Petri nets, their parallel composition is represented by

the disjoint union of these nets. In the case of workflow nets, this representation leads to a proliferation of input and output places. Therefore we will consider workflow nets with multiple pairs of input and output places. These will also be suitable to represent business processes that can deal with different types of input that are treated in different ways and lead to different types of output.

A *multi-organizational workflow net* $W = (S, T, F, \lambda, IO)$ is a predicate/transition net (S, T, F, λ) equipped with a set IO of *input/output ports*, each port $p \in IO$ consisting of an *input place* $p_{in} \in S$ and an *output place* $p_{out} \in S$.

Each of the two workflows of Fig. 1, not including the dashed arcs between them, can be regarded as a multi-organizational workflow net. The dealership's workflow has two input/output ports, namely "client" and "car (or question on repair car)".

1.7 Workflow Modules

We define a *workflow module* $(S, T, F, \lambda, IO, Q)$ to be a multi-organizational workflow net equipped with a set Q of *query ports*. Each query port $q \in Q$ consists of two transitions $q_?$ and $q_! \in T$ and two places q_{pre} and $q_{active} \in S$, connected by arcs $(q_{pre}, x_1, q_?), (q_?, x_2, q_{active})$ and $(q_{active}, x_3, q_!) \in F$. There are no other arcs to or from q_{active}. The arcs leading to q_{pre} should be such that q_{pre} receives at most one token for each input received by the workflow module. In some cases this is achieved by allowing only arcs to q_{pre} from transitions that have an input place p_{in} with $p \in IO$ as preplace (Fig. 2).

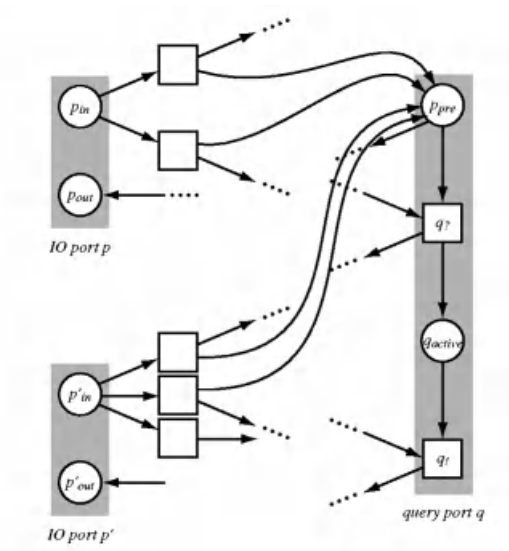

Fig. 2. *A query port in a workflow module. The query port q consists of a query transition $q_?$ sending a query to another workflow module, a query transition $q_!$ receiving the answer from the other module, a query place q_{pre} ensuring that in each run of the workflow module the query transition $q_?$ can be fired at most once, and a query place q_{active} ensuring that $q_?$ fires before $q_!$.*

Workflow modules represent business processes with the ability to invoke other business processes as subroutines. Each query port $q \in Q$ models (a) the invocation of another business process (the *target process* of the port) by sending it an input, and (b) the receipt of the corresponding output. The query transition $q_?$ represents the act of invoking the target process. The transition guard $\lambda(q_?)$ may contain free occurrences of the variable $x_?$. This variable represents the input that is presented to the target process. The query transition $q_!$ represents the receipt of an output from the target process. The transition guard $\lambda(q_!)$ may contain free occurrences of the variable $x_!$, referring to that output. This guard may relate $x_!$ with variables labeling outgoing arcs of $q_!$; however, it is not allowed to restrict the enabling of $q_!$ based on the value of $x_!$.

The query place q_{pre} ensures that in each run of the workflow started by single token in an input place, the query transition $q_?$ can fire at most once. In order to model a business process that in one such run invokes another business process twice as a subroutine, one needs two different query ports. When the target process has been invoked, but its output has not yet been collected, the query place q_{active} contains a token. This design ensures that $q_!$ can fire only after the firing of $q_?$.

1.8 Modular Workflow Architectures

A *modular workflow architecture* is a finite set of workflow modules, together with an allocation, to each query port in each module in the architecture, of an input/output port of another (or the same) workflow module (Fig. 3). The input/output port allocated to a query port is called the *target* of that query port. We assume that query ports in different modules in the architecture have different names.

A modular workflow architecture models a collection of interconnected business processes. Each of these business processes is represented by a workflow module. Occasionally such a business process delegates a subtask to another business process. This delegation is modeled by the query ports. The target of a query port indicates to which business process the subtask is delegated. A business process may dynamically choose a module to which a task should be delegated (e.g. in vendor selection) by routing control through a chosen query port.

A modular network architecture can be represented by a single multi-organizational workflow net, called the *cross-organizational workflow net* representing the architecture. This is done by taking the disjoint union of its constituent workflow modules, and connecting every query port q in every module, through arcs $(q_?, x_?, p_{in})$ and $(p_{out}, x_!, q_!)$, with the input and output places p_{in} and p_{out} of the target port p of q (Fig. 4). The input/output ports of the cross-organizational workflow net are those of the constituent modules. Figure 1 shows a modular workflow architecture that is made into a cross-organizational workflow net by adding the dashed arcs.

2 Workflow Nets for Multiple Cases

A *case* is a run of a business process invoked by a single input. In a workflow net a case starts when a token is deposited by the environment in the input place of the net, and ends when a token arrives at the output place. In van der Aalst [1,2], workflows are considered to be *case driven*, meaning that every case can be considered as running in

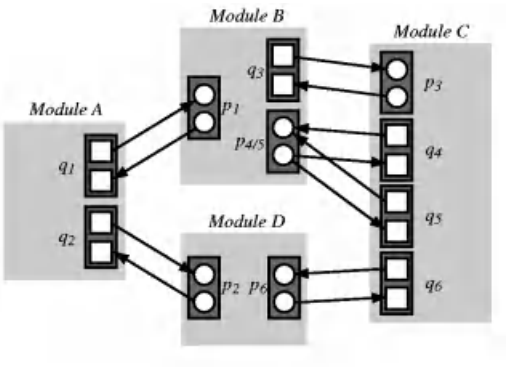

Fig. 3. *A modular workflow architecture consisting of four workflow modules with six query ports. The query ports q_4 and q_5 share the same target $p_{4/5}$.*

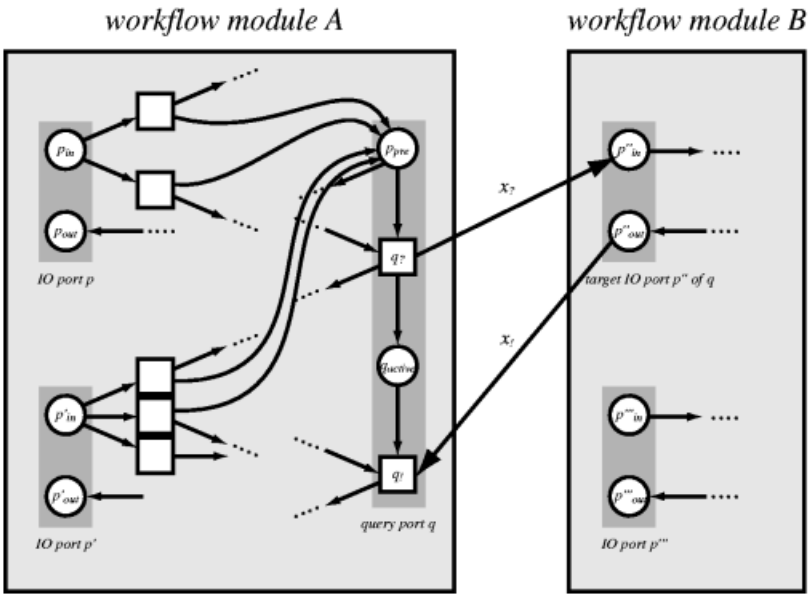

Fig. 4. *A query port in a workflow module and its connection in a cross-organizational workflow net to the target of that port.*

a fresh copy of the workflow net. This approach guarantees that different cases do not influence each other, a property we call *case independence*.

The case driven approach would unduly complicate the constructions of the present paper. We consider different cases to be running in parallel in the same workflow net, which can happen in particular in workflow modules within modular workflow architectures, namely when a module is invoked twice as a subroutine in the course of executing

a single case of another module. Nevertheless, case independence can be ensured easily in predicate/transition nets by assigning a unique identifier to every case. In practice, this identifier could consist of the name and address of whomever presented the case to the workflow, and the time of submission. To this end we assume that every token deposited in the input place of a workflow net carries a unique case identifier. The transition guards of each transition will require that that transition fires only when all incoming tokens have the same identifier. Each output token will have that identifier as well. This method precludes any interference between cases.

In this section, we first formally define case independence, and then describe a method to transform any multi-organizational workflow net into a case independent one under the assumption made above. Subsequently we extend this method to workflow modules in such a way that the case-independence of modules is preserved when these modules are embedded in a cross-organizational workflow net.

2.1 Case Independence

A multi-organizational workflow net W enjoys *case independence* if any firing sequence $M_0[t_1\rangle M_1[t_2\rangle M_2[t_3\rangle \cdots$ where M_0 is a marking that only puts tokens in input places of W, can be obtained by *interleaving* firing sequences that start with markings that put only one token in an input place. This means that, if the initial marking M_0 has n tokens, each marking M_i can be written as $M_{i1} + \cdots + M_{in}$, such that, for $j = 1, ..., n$, M_{0j} puts only a single token in an input place, and for each transition t_i in the sequence, there is a $j \in \{1, ..., n\}$ such that $M_{(i-1)j}[t_j\rangle M_{ij}$ and $M_{(i-1)k} = M_{ik}$ for $k \neq j$. In words, the given firing sequence can be decomposed into n firing sequences starting with a 1-token marking, and every transition t_i belongs to one of these n firing sequences.

We do not aim for general case independence, but merely for case independence under the assumption that the environment supplies tokens that are tagged with distinct identifiers from a set ID. We call this *ID-case independence*. It is formally defined just as case independence above, but only for initial markings M_0 that supply tokens of the form (id, d) with $id \in ID$ such that all supplied tokens have distinct identifiers.

2.2 Ensuring Case Independence

Let $W = (S, T, F, \lambda, IO)$ be a multi-organizational workflow net designed for the case driven approach, in a framework with $\mathcal{D}$ the set of possible tokens. Let ***ID*** be a set of possible identifiers, and take $\mathcal{D}' = \mathit{ID} \times \mathcal{D} = \{(id, d) \mid id \in \mathit{ID} \wedge d \in \mathcal{D}\}$, the set of possible tokens with attached identifiers. Suppose $\{x_1, \ldots, x_n\}$ is the set of variables labeling the arcs entering and leaving a transition $t \in T$, then we define $\lambda'(t)$ to be

$$\exists\, id \in \mathit{ID}\colon x_1 = (id, x_1') \wedge \cdots \wedge x_n = (id, x_n') \wedge \lambda(t)[x_i'/x_i\ (i = 1, ..., n)]$$

where $\lambda(t)[x_i'/x_i\ (i = 1, ..., n)]$ denotes the result of substituting x_i' for x_i in $\lambda(t)$ for $i = 1, ..., n$. Now $W' = (S, T, F, \lambda', IO)$ is a multi-organizational workflow net that behaves just like W when presented with a single input token, but in which ID-case independence is guaranteed. The framework in which W' is specified uses the domain $\mathcal{D}'$ of possible tokens and a suitably extended collection $\mathbb{F}'$ of formulas.

2.3 Ensuring Case Independence in Modular Workflow Architectures

We consider a modular workflow architecture to be ID-case independent if its individual modules are case independent under the assumption that tokens deposited by the environment of the architecture in input places of the modules of the architecture have distinct identifiers, chosen from the set ID. For our main results below, ID-case independence of modular workflow architectures is essential. ID-case independence of modular workflow architectures can be achieved by the method described in Sect. 2.2, provided that all tokens deposited in input places of the modules of the architecture have distinct identifiers. Such tokens may be provided by the environment or by query transitions. Therefore we have to require that query transitions never emit tokens with identical identifiers to input ports of any given module. This requirement will be fulfilled when query ports augment the identifier of a case they are dealing with with their own identity. To this end, for a collection of identifiers ID supplied by the environment, and a collection of $\mathbf{Q}$ of potential names of query ports appearing in a modular workflow architectures, let $ID_{\mathbf{Q}}$ be the collection of identifiers $id * q_1 * q_2 * \cdots * q_k$ with $k \in \mathbb{N}$, $id \in ID$ and $q_j \in \mathbf{Q}$ for $j = 1, ..., k$. Such identifiers consist of a global case identifier id from ID, attached to a token by the environment, together with a sequence of names of query ports that the token passed through in succession. For a query port $q \in \mathbf{Q}$ the transition guards $\lambda'(q_?)$ and $\lambda'(q_!)$ now are $\exists\, id \in ID_Q : x_1 = (id, x'_1) \wedge \cdots \wedge x_n = (id, x'_n) \wedge x_? = (id{*}q, x'_?) \wedge \lambda(q_?)[x'_i/x_i\ (i{=}1, ..., n, ?)]$ and $\exists\, id \in ID_Q : x_1 = (id, x'_1) \wedge \cdots \wedge x_n = (id, x'_n) \wedge x_! = (id{*}q, x'_!) \wedge \lambda(q_!)[x'_i/x_i\ (i{=}1, \ldots, n, !)]$. It could be that in a single case a single module invokes another module twice by means of different query ports. Using the names of those ports to augment the case identifier results in the invoked module only receiving tokens with distinct identifiers. Here it is crucial that in a single case any particular query port q be used only once. For this reason we introduced the query places q_{pre}.

We call a workflow module *manifestly ID-case independent* if it has been made *ID*-case independent by the method above. If all workflow modules in a modular workflow architecture are manifestly *ID*-case independent, then surely the architecture itself is *ID*-case independent.

In a modular workflow architecture consisting of manifestly ID-case independent workflow modules the type of token identifiers is that of a stack. Tokens deposited by the environment are assumed to have an identifier $id \in ID$. Whenever a token passes though a query transition $q_?$ into another module (containing the target of q) the name q of the corresponding query port is appended to the identifier, or "pushed on the token's stack". Within a module the identifier of tokens is preserved. When a token is retrieved by the query transition $q_!$ from the output place of the target of q, the name q is popped from the token's stack.

3 Proper Termination

Typically, van der Aalst [1,2] requires workflow nets to be *sound*, in the sense that
(1) if a token is put in the input place, a token will eventually appear in the output place,
(2) when a token appears in the output place, no other tokens are left in the net, and
(3) every transition in the net can under some circumstance be fired.

These properties are formalized as follows (where the marking $\{\!|i|\!\}$ is the multiset that only contains the input place of the workflow net):

(1) Any fair firing sequence starting with $\{\!|i|\!\}$ contains a marking with a token in o.
(2) Any marking that is reachable from $\{\!|i|\!\}$ and has a token in o, must equal $\{\!|o|\!\}$.
(3) For any transition t there is a firing sequence starting with $\{\!|i|\!\}$ in which t appears.

In van der Aalst [1,2] workflow nets are required to satisfy

(4^*) The input place i does not have incoming arcs and the output place o does not have outgoing arcs. Furthermore, for every place or transition $r \in S \cup T$ there should be a path in the net from i to o via r.

This structural property implies that (4) in the marking $\{\!|o|\!\}$ no transition is enabled. In this paper, we will drop the structural requirement (4^*) but retain its consequence (4), which will be treated as an additional soundness requirement.

Soundness property (1) says that the workflow net is guaranteed to provide an output token, and property (2) adds that when the environment retrieves this token from the output place, no tokens remain in the net. In combination with the assumed case independence of workflow nets, properties (1) and (2) imply that for each token put in the input place, exactly one token will eventually appear in the output place. It is not hard to find counterexamples showing that this does not follow without case independence.

Soundness property (3) is one of parsimony, and has no bearing on the operational behavior of the workflow net. Any workflow net can be transformed into an operationally equivalent one that satisfies this property, namely by deleting all transitions that can never be fired. We will not be concerned with this soundness property here, and instead use a concept of soundness consisting of properties (1), (2), and (4). This form of soundness is called *proper termination* [4].

The following definition extends this concept to multi-organizational workflow nets for which all tokens are of the form (id, d) with id a case identifier. From here onwards we work in a framework for predicate/transition nets in which all token have this form. Now a marking is a multiset of elements $(s, (id, d))$ with s a place in the net. As the case identifier is supplied by the environment dropping a token in an input place, and that environment is hoping for an output token pertaining to the same case, only tokens of the form (id, d') count as legitimate output when an input (id, d) was supplied. Tokens with any other identifier may pass through output places casually.

Definition 1. *A multi-organizational workflow net is* properly terminating *if*

(1) Any fair firing sequence starting with a marking $\{\!|(p_{in}, (id, d))|\!\}$ for some input/output port $p \in IO$ contains a marking with a token (id, d') in p_{out}.
(2) Any marking that is reachable from a marking $\{\!|(p_{in}, (id, d))|\!\}$ and has a token (id, d') in p_{out}, must equal $\{\!|(p_{out}, (id, d'))|\!\}$.
(4) In a marking $\{\!|(p_{out}, (id, d))|\!\}$ no transition is enabled.

Note that because in a workflow module in a modular workflow architecture a query transition $q_!$ lacks its incoming arc $(p_{out}, x_!, q_!)$ where p is the target of q, proper termination of the workflow module can be understood as proper termination of the workflow module enriched with such an arc, under the assumption that a token is supplied over this arc, i.e., under the assumption that the workflow invoked by $q_?$ terminates. This is what we called *local termination* in the introduction. Also note that, in view of the firing rule for predicate/transition nets, proper termination is required for *any* token supplied over this arc, i.e., for any output returned by the invoked workflow.

3.1 Proper Termination of Modular Workflow Architectures

We consider a modular workflow architecture to be properly terminating whenever the cross-organizational workflow net representing the architecture is properly terminating. The main goal of this paper is to formulate conditions on workflow modules in an architecture that guarantee proper termination of the architecture itself.

Lemma 1. *If all workflow modules in a modular workflow architecture satisfy property (4) of Definition 1, then the cross-organizational workflow net W representing the architecture satisfies property (4).*

Proof. Let M be a marking $\{\!\{(p_{out}, (id, d))\}\!\}$ of W. As in each of the queries q in W the place q_{active} is not marked, none of the query transitions $q_!$ is enabled under M. Any other transition t in W is enabled under M only if in the module containing t, t is enabled under the restriction of M to that module, which is either $\{\!\{(p_{out}, (id, d))\}\!\}$ or the empty marking. As that module satisfies (4), t is not enabled under $\{\!\{(p_{out}, (id, d))\}\!\}$, and thus certainly not under the empty marking.

Lemma 2. *If all workflow modules in a manifestly ID-case independent modular workflow architecture satisfy properties (2,4) of Definition 1, then the cross-organizational workflow net W representing the architecture satisfies property (2).*

Proof. Let σ be a firing sequence in W starting with $\{\!\{(p^0_{in}, (id_0, d_0))\}\!\}$, and let M be a marking in σ with a token (id_0, d'_0) in p^0_{out}. As W is manifestly *ID*-case independent, each token occurring in a marking in σ has a case identifier of the form $id_0{*}q_1{*}q_2{*}\cdots{*}q_k$. Here we also use that in W there are no transitions without incoming arcs, which follows from the assumption that W satisfies (4). Furthermore, each transition in σ has a transition guard of the form $\exists id \in ID_Q\colon\ \varphi$ and therefore can be annotated with the identifier $id \in ID_Q$ that enabled it. For any $id \in ID_Q$ and any marking M' let $M' \restriction id$ consist of the elements $(s, (id, d))$ of M, and let $\sigma \restriction id$ be obtained from σ by replacing its markings M' by $M' \restriction id$, and by skipping the transitions that are not annotated with id. From the manifest ID-case independence of W it follows that $\sigma \restriction id$ is a firing sequence in one of the modules (that we call W_{id}). It follows immediately from the assumption that W_{id_0} satisfies (2) that $M \restriction id_0 = \{\!\{(p^0_{out}, (id_0, d'_0))\}\!\}$.

With induction on $k > 0$, we establish that $M \restriction id_0{*}q_1{*}q_2{*}\cdots{*}q_k$ is empty, which finishes the argument. So assume $M \restriction id$ is either $\{\!\{(p^0_{out}, (id_0, d'_0))\}\!\}$ or empty. It has to be shown that $M \restriction id{*}q$ is empty. In case $\sigma \restriction id$ does not contain the transition $q_?$ this is trivial. In case $\sigma \restriction id$ does contain $q_?$, it must also contain $q_!$, as $M \restriction id$ has no tokens in q_{active}. Hence $\sigma \restriction id{*}q$ starts with a marking $\{\!\{(p_{in}, (id{*}q, d))\}\!\}$ and has a marking M' with a token $(id{*}q, d')$ in p_{out}, where p is the target of q. As $W_{id{*}q}$ satisfies (2), $M' \restriction id{*}q = \{\!\{(p_{out}, (id, d'))\}\!\}$, and $M \restriction id{*}q$ must be empty.

It is not hard to see that a modular workflow architecture may fail to be properly terminating if it fails to be *ID*-case independent, or if some its workflow modules fail to be properly terminating. However, (manifest) ID-case independence and proper termination of the workflow modules in a modular workflow architecture are not sufficient conditions to guarantee proper termination of a modular workflow architecture. A failure of proper termination of a modular workflow architecture may occur in the case of loops in the connections between its workflow modules. If, for example, module A keeps calling module B and vice versa, as can happen in the car dealership example from the

introduction, the resulting architecture has a loop and the associated queries will never be answered.

Below we define a subclass of *acyclic* modular workflow architectures for which the requirements that its modules are manifestly *ID*-case independent and properly terminating are sufficient to ensure proper termination of the architecture. In Sect. 3.3 we will show that the condition of acyclicality can be omitted when equipping workflow modules with a simple business rule that, in essence, prohibits the posing of the same query twice. Workflow modules that are so equipped will be called *query nets*.

3.2 Acyclic Architectures

The *connectivity graph* of a modular workflow architecture has as its nodes the workflow modules in the architecture, and a directed edge $A \to B$ whenever workflow module A has one or more query ports with a target in B. Figure 5 shows the connectivity graph of the modular workflow architecture of Fig. 3. An architecture is called *acyclic* if its connectivity graph has no cycles.

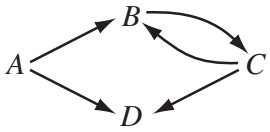

Fig. 5. *Connectivity graph of the modular workflow architecture of Fig. 3. The letters represent workflow modules and the arcs represent queries. This architecture is cyclic because module B contains a query port with a target in module C and vice versa.*

Theorem 1. *If all workflow modules in an acyclic modular workflow architecture are manifestly ID-case independent and properly terminating, then the architecture itself is properly terminating.*

Proof. That the cross-organizational workflow net W representing the architecture satisfies properties (4) and (2) of Definition 1 follows from Lemma's 1 and 2. Now consider a fair firing sequence σ in W starting with a marking $\{\!|(p_{in}, (id, d))|\!\}$. Using the notation and results from the proof of Lemma 2, $\sigma \restriction id$ is a firing sequence of the module W_{id}. In case this firing sequence is fair, as W_{id} satisfies (1), $\sigma \restriction id$, and hence also σ, must contain a marking with a token (id, d') in p_{out}, which had to be established. The only way $\sigma \restriction id$ can fail to be fair in W_{id}, even though σ is fair in W, is when there is a query transition $q_!$ in W_{id} that, from a certain marking in $\sigma \restriction id$ onwards, is continuously enabled but never fires. In σ this query transition cannot be continuously enabled, which is possible only when $\sigma \restriction id{*}q$ is a firing sequence in W_{id*q} starting with a marking $\{\!|(p^1_{in}, (id{*}q, d_1))|\!\}$ but having no marking with a token $(id{*}q, d'_1)$ in the output place p^1_{out} of the target p^1 of q. (If a such token does arrive in p^1_{out} and $q_!$ never fires, that token is stuck in p^1_{out} by properties (2) and (4) of W_{id*q}, contradicting the fairness of σ.) As W_{id*q} satisfies (1), $\sigma \restriction id{*}q$ cannot be fair, even though σ is. This can only be explained

by a query transition $q'_!$ in W_{id*q}, with similar properties as $q_!$ above. Continuing in this vein, we find an infinite sequence of query transitions $q_?$ visited by σ, contradicting the acyclicality of the architecture.

3.3 Query Nets

A *query net* is a properly terminating manifestly *ID*-case independent workflow module that never poses the same query twice. The latter can be achieved by a clause in the transition guards of query transitions $q_?$ forbidding the transition to fire when the token identifier contains the name q of that query already. Thus a query net implements a business rule that prevents getting stuck in an infinite loop. The requirement of proper termination moreover implies that the workflow module should embody a backup plan to deal with the situation that the interaction with other workflow modules would have given rise to such a loop. This backup plan may involve a transition that can fire as an alternative to $q_?$ when $q_?$ appears in the identifier of an token. Such an alternative transition must have q_{pre} as one of its input places. An example of this is the transition "admit car cannot be repaired" in Fig. 1. We can now state our main result.

Theorem 2. *If all workflow modules in a modular workflow architecture are query nets, then the architecture is properly terminating.*

Proof. Exactly as for Theorem 1, but this time an infinite sequence of query transitions $q_?$ without matching $q_!$'s cannot be visited, because there are only finitely many queries in the architecture, and no query can occur twice in the sequence.

In fact, Theorems 1 and 2 can be combined and strengthened by merely requiring that all workflow modules in the modular workflow architecture are manifestly *ID*-case independent and properly terminating , and that in any cycle in the connectivity graph of the architecture there is at least one query net.

The dealership's workflow of Fig. 1 is, for adequate choices of the transition guards λ_i, a query net. The manufacturer's workflow on the other hand is not, as its query is not equipped with a backup plan. Nevertheless, the cross-organizational workflow net that combines both modules is, for adequate choices of the λ_i's, properly terminating.

4 Conclusion and Comparison with Related Work

Petri nets have been established as a powerful model for workflow applications. Van der Aalst [1,2] has proposed *soundness criteria* that guarantee that in a business application modeled by a workflow net every case submitted to the workflow will be completed, and with no references to it remaining in the net. In this paper we examined cross-organizational business applications that are modeled by collections of communicating workflow nets. The amalgamation of all these workflows into a single workflow net may be too large for possibly automated formal analysis. Moreover, individual business partners that operate one of the workflows in the collection may be reluctant to provide the complete specification of their workflow, as result of which nobody can know the complete specification of the amalgamated workflow. For this reason, we explored ways to establish global termination properties for the amalgamated workflow by investigating whether local termination properties hold for the individual workflows in the collection.

We proposed local properties that can be checked for individual workflow nets in the collection (by the organizations that operate these individual workflows), without the need for any knowledge of the other workflows in the collection. These local properties guarantee global termination of the amalgamation.

Addressing the same issue, Kindler, Martens and Reisig [6] establish that global termination of the amalgamated workflow is implied by local termination of the component workflows, provided those components are *locally correct with respect to a fairness-closed specification*. In fact, their definition of a *fairness-closed specification* is carefully crafted in such a way that this result holds. In many realistic applications global termination fails even when local termination of the component workflows holds (see Section 1.4 in [6]). It turns out that in such cases there is no fairness-closed specification for which the component workflows are locally correct. Thus these specifications are essential. A problem is that fairness-closed specifications specify the interactions between all workflow modules in the amalgamated workflow. Thus checking correctness of a workflow module with respect to such a specification requires more than local knowledge about that module.

In our approach there is no need for such fairness-closed specifications. Instead, for each of the modules we check locally that a simple business rule is obeyed, that in essence prohibits asking the same question twice. In this way we ensure global termination by checking local properties only.

References

1. Wil M. P. van der Aalst (1999): *Interorganizational Workflows: An Approach Based on Message Sequence Charts and Petri Nets. Systems Analysis—Modelling—Simulation* 34(3), pp. 335–367.
2. Wil M. P. van der Aalst & Kees M. van Hee (2002): *Workflow Management: Models, Methods, and Systems.* MIT Press.
3. Hartmann J. Genrich (1987): *Predicate/Transition nets.* In Wilfried Brauer, Wolfgang Reisig & Grzegorz Rozenberg, editors: *Petri nets: Central Models and Their Properties*, Advances in Petri nets 1986, Part I, LNCS 254, Springer, pp. 207–247.
4. K. Gostellow, V. Cerf, G. Estrin & S. Volansky (1972): *Proper Termination of Flow-of-control in Programs Involving Concurrent Processes.* ACM Sigplan $7^{(11)}$, pp. 15–27.
5. Kurt Jensen (1994): *An Introduction to the Theoretical Aspects of Coloured Petri Nets.* In Jaco W. de Bakker, Willem-Paul de Roever & Grzegorz Rozenberg, editors: *A Decade of Concurrency*, LNCS 803, Springer, pp. 230–272.
   Available from `http://www.daimi.au.dk/~kjensen/papers_books/rex.pdf`.
6. Ekkart Kindler, Axel Martens & Wolfgang Reisig (2000): *Inter-operability of Workflow Applications: Local Criteria for Global Soundness.* In Wil van der Aalst et al., editor: *Business Process Management*, LNCS 1806, Springer, pp. 235–253.
7. Einar Smith (1998): *Principles of High-level Petri Nets.* In Wolfgang Reisig & Grzegorz Rozenberg, editors: *Lectures on Petri nets I: Basic models*, *Advances in Petri nets*, LNCS 1491, Springer, pp. 174–210.
8. Workflow Management Coalition (1995): *The Workflow Reference Model.* Available from `http://www.wfmc.org/`.

Generic Recurrent Patterns in Business Processes

Jan L.G. Dietz

Delft University of Technology
P.O. Box 5031, NL-2600GA Delft
j.l.g.dietz@its.tudelft.nl

Abstract. There doesn't seem to be much commonality among business processes, even not if they belong to the same kind of organization. However, by applying the right kind of abstraction from realization issues and by rooting this abstraction is the CAP-theory, it appears that there is a generic recurrent pattern in all business processes. This pattern, called the transaction, is presented and elaborated in this paper. The part of the underlying CAP-theory (Coordination-Actors-Production) that is necessary for understanding and appreciating it is explained. The focus in this paper is on the Coordination aspect. An outlook is given on the potential benefits of the transaction pattern for the analysis and design of business processes.

1 Introduction

The business processes in different organizations seem to be very different, even in organizations of the same kind. At least, one gets this impression when surveying the divergent models process analysts make of the processes in similar organizations. It looks as if every business process is a unique sequence of steps that just happens to be as it is, or that just happens to be conceived so by modelers. However, business processes are not as unique and as arbitrary as they seem to be at first sight. Over ten years of practical experience with the DEMO[1] methodology has demonstrated that the business processes in all organizations display generic recurrent patterns. These patterns are universal in the sense that they do not depend on the kind of organization. Otherwise said, they occur equally in manufacturing companies, service companies, governmental agencies, shops, unions etc. Much like physical things are governed by the laws of physics, the 'business' behavior of people obeys socionomic laws. It is the purpose of this paper to provide both theoretical and practical evidence for the existence of these laws, and to elucidate the benefits of using the existing socionomic patterns for the modeling, analysis, design, and engineering of business processes[2].

[1] DEMO is an acronym for 'Demo Engineering Methodology for Organizations'. The methodology is primarily meant to demonstrate that Organization Engineering is a sensible and valuable concept, and that a sound theoretical basis can be provided for it.

[2] The constraints on the size of the paper prohibit to explain everything at the desired level of detail. A full account is given in the homonymic publication on www.demo.nl

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 200–215, 2003.

An organization can be studied from a functional or from a constructional perspective. Taking the *functional* perspective means that one looks at the function of an organization with respect to its environment, and at the behavior it displays when functioning. For example, a hospital is an organization of which the (primary) function is to provide health care to people. Its behavior can be described in terms of the number of patients that are treated per period, the distribution of the treatment duration etc. A model of an organization from the functional perspective is called a *black-box* model. The functional perspective is typically taken by managers and administrators. A well-known kind of black-box model is the value chain [22]. Taking the *constructional* perspective on an organization means that one wants to understand how its function and behavior are brought about by its construction and operation. The models used to study organizations in this perspective are called *white-box* models. As will be shown in the next sections, the operating principle of an organization consists of the ability of human beings to enter into and comply with commitments towards each other. The constructional perspective on organizations has not so long a tradition as the functional one has. Although this perspective would be the right one for them, in practice few designers of business processes and workflows appear and few developers of ICT-applications do take it.

From the constructional perspective, an organization may be modeled as a *discrete dynamic system* in the category of *social systems* [7], which has the next properties. Its composition consists of active elements, i.e. elements that possess the ability to perform actions. Each of these elements is defined by the specific set of actions it is able to execute. They are activated by the actions of (other) active elements. At any moment, the system is in a particular state. State transitions take place as the result of actions of the active elements. They occur at discrete points in time and take place instantaneously [15]. Conceiving an organization as a discrete dynamic system, means that the members of the organization take the role of active elements of the system. This may be justified by considering that the people in an organization do have their own specific tasks and that they collaborate (in mutual interaction) in order to bring about the function of the organization. It also means that one disregards the possibly continuous character of the operation of an organization by only taking the state of the system into account at particular time intervals. Generally spoken, this is justified by considering that for the purpose of controlling the (business) processes in the organization, it is sufficient to know the state and the progress of these processes at discrete points in time. The same consideration justifies that state changes take place instantaneously.

The outline of the paper is as follows. In section 2, the theory behind the DEMO methodology is presented as far as it is needed for the remainder of the paper. In section 3, the generic recurrent pattern of a transaction is introduced and elaborated. The universal applicability of this pattern is demonstrated in section 4, taking a pizzeria as the example case. Several DEMO-models are presented to illustrate the practical application of the transaction concept. In section 5, the research outcomes are discussed, in comparison with other contemporary approaches that model organizations as discrete dynamic systems. It also contains the conclusions that can be drawn, and some suggestions for future research.

2 The CAP-Theory

In this section, a brief overview is given of the so-called CAP-theory, which underlies the DEMO methodology [11, 12, 13]. It is a theory about the construction and operation of organizations. The name 'CAP' stands for Coordination, Actors and Production. These are the primary concepts for understanding the operation of organizations and for analyzing and designing their business processes. Following the distinction of system categories as proposed by Bunge [7], an organization is a *social system*, i.e. a system of which the elements are social individuals or *subjects*. *Organizations* are a specialization of social systems, the distinctive property being that they are *designed* to serve a particular *purpose*. This purpose can generally be defined as *delivering products* to its environment. The products may be material (goods) or immaterial (services). The bringing about of products is where the *production* part of CAP stands for. Products are brought about by subjects acting in particular actor roles. This is where the *actors* part of CAP stands for. The actors co-operate, i.e. they activate each other in order to collectively realize the Production. This is where the *coordination* part of CAP stands for. In this paper, the focus is on the coordination, since this aspect is the most important one for understanding the generic recurrent patterns.

Stated more specifically, the subjects in an organization perform two kinds of acts: production acts (or P-acts for short) and coordination acts (or C-acts for short). By performing *production acts*, they contribute to bringing about the goods and/or services that are delivered to the environment of the organization. The character of a production act is material or immaterial. Examples of material acts are all manufacturing acts as well as storage and transportation acts of goods. Examples of immaterial acts are the judgment by a court to condemn someone, the decision to grant an insurance claim, and appointing someone president. By performing *coordination acts*, subjects enter into and comply with commitments and agreements towards each other regarding the performance of production acts. The notion of coordination is taken broadly; it encompasses all interaction between subjects in an organization as well as between them and subjects in the environment.

In order to abstract from the particular subject that performs an action and to concentrate on the organizational role of the subject in performing that action, the notion of *actor role* is introduced. It is defined as the 'amount' of authority to perform particular acts. An actor role may be played by several subjects and a subject may play several actor roles. An *actor* is a subject in his/her fulfillment of an actor role.

The result of successfully performing a production act is a *production fact* or P-fact (cf. figure 1). Examples of production facts (in the context of a library) are "membership M has started to exist" and "the fine for loan L has been paid". The variables M and L denote particular instances of the entity type membership and loan respectively. Examples of coordination acts are the requesting and the promising of a production fact, e.g. requesting to become member of the library. The result of successfully performing a coordination act is a *coordination fact* or C-fact. An

example of a coordination fact is the being requested of the production fact "membership #387 has started to exist".

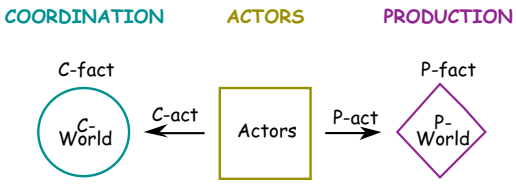

Fig. 1. The operational model of an organization

In conformity with the distinction between production acts and coordination acts, two worlds can be distinguished in which each of these kinds of acts have effect: the *production world* or P-world and the *coordination world* or C-world respectively. A state of the P-world is a set of P-facts, and a state of the C-world is a set of C-facts. The creation of the P-fact "membership #387 has started to exist" is a state *transition* in the P-world. The creation of the C-fact "(membership #387 has started to exist) is requested" is a state *transition* in the C-world. A particular transition at a particular point in time (e.g. creating the P-fact "membership #387 has started to exist") is an *event*.

2.1 Coordination

Coordination takes place in communication. Research in the so-called Language Action Perspective (LAP) has shed a new, clarifying light on human communication [4, 9, 10, 14, 28]. In order to cope adequately with the coordination part of the CAP-theory, the major findings of the LAP research have been adopted in the CAP-theory as the *Communicative Action Principle* (C-CAP for short). According to the C-CAP, a coordination act is performed by one actor (called the *performer*) and directed to another actor (called the *addressee*). The generic structure of a coordination act is exhibited in figure 2.

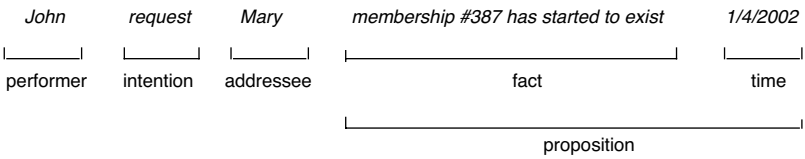

Fig. 2. The structure of a coordination act

The example concerns the request by the subject John towards the subject Mary to become member of the library. Every new case of a person becoming member, is conceived as the creation of a new instance of the entity type membership. In the

example, the instance created is #387, of which John is the member. The *proposition* in a coordination act consists of a fact and an associated time. The *fact* is a production fact. The *time* attribute refers to the time period in which the fact is the case (in which it is said to exist). In the example of figure 3, its meaning is the requested starting date of the membership. The *intention* of a coordination act represents the 'social attitude' taken by the performer with respect to the proposition. The term is borrowed from Searle [25] and we use it to distinguish coordination acts from communicative acts where the term 'illocution' is commonly used to refer to the 'attitude' of the speaker [14, 24]. Examples of intentions in DEMO are: request, promise, state and accept. They belong to Habermas' category of regulativa [9, 14].

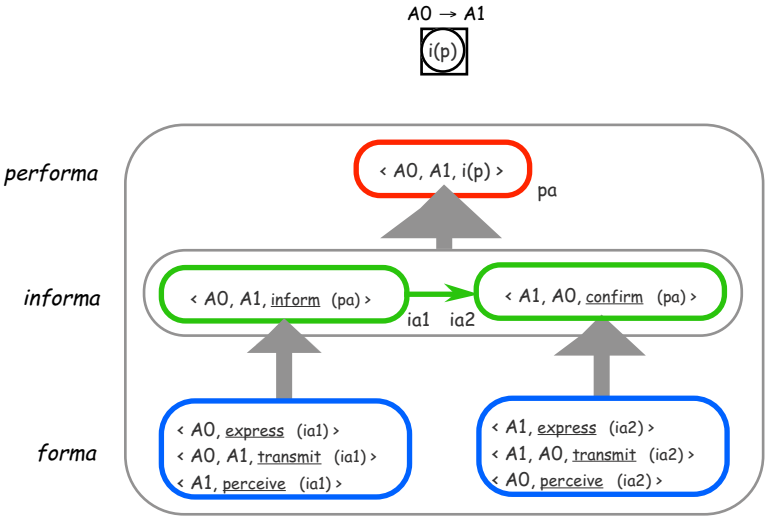

Fig. 3. The process of performing a coordination act

A coordination act is brought about in a sequence of communicative acts, that goes through all three layers of communication (cf. [13]). Figure 3 exhibits the three kinds of acts that have to be executed: performative, informative, and formative acts. The combined box and disk at the top of the figure represents both the coordination act (the box) and the resulting coordination fact (the disk). They are identified by 'i(p)', where 'i' is the intention and 'p' the proposition. Actor A0 is the performer of the coordination act and A1 is the addressee. There are three conditions that must be satisfied for successfully performing a coordination act: the performa-condition, the informa-condition and the forma-condition. By satisfying the *performa-condition* is meant that the correct social understanding is raised in Mary. In the given example it is the understanding of being committed to respond in a socially appropriate way to the request (intention i) by John regarding him becoming member of the library (proposition p). The performative act, noted as < A0, A1, i(p) > succeeds if this performa-condition is satisfied. A necessary precondition for the performa-condition is the *informa-condition*, by which is meant the establishment of the intellectual understanding of the coordination act by Mary. Satisfying this condition is usually

achieved through so-called *informative exchanges*. Basically, such an exchange consists of the informative acts < A0, A1, inform (pa) > and < A0, A1, confirm (pa)>, but it may also contain a number of question-assertion pairs in which the intellectual understanding is clarified. They belong to Habermas' category of constativa [9, 14].The *forma-condition* concerns the establishment of the linguistic understanding of the coordination act. It is satisfied if there is a well-functioning communication channel between John and Mary, through which they are able to perform the formative acts that realize the informative acts. For example, the informative act, noted as < A0, A1, inform(pa) > and referred to as 'ia1', consists at the forma-layer of expressing the content in a sentence by John, transmitting the sentence from John to Mary, and perceiving the sentence by Mary.

2.2 Actors

In most discrete dynamic system approaches, a simple trigger-action or event-response principle is adopted for modeling the operation of the active elements. This is perfectly well for physical, chemical and biological systems and even for rational systems like all information systems, as applied e.g. in [15] or in Petri Net Models [2, 21]. However, such a principle is too mechanistic for social systems, it wouldn't do justice to the social character of human beings. Therefore the *Committed Agenda Principle* (A-CAP for short) is introduced to explain the 'operation' of actors. It states that at every moment an actor has at his disposal a set of actions to take, called his *agenda*. An agenda-item or agendum is a C-fact to which the actor has *committed* himself to respond. For example (cf. figure 2), if John has successfully requested Mary to become member of the library, the created C-fact "(membership #387 has started to exist) is requested" has become an agendum for Mary. She has to deal with it in a responsible way, i.e. to respond socially adequate, which includes acting in time. For every kind of agendum, there is a particular *action rule* that serves as a guideline for dealing with the agendum. Action rules in DEMO are similar to the ECAA rules as discussed in [17].

2.3 Production

Performing a production act (P-act) means creating a production fact (P-fact), thereby causing the corresponding transition of the state of the P-world to occur. By applying the *Core Activity Principle* (P-CAP for short), one can distinguish the *essential* business acts from supporting acts. To find the essential acts and facts, one starts from the so-called *independent* transactions. These are the transactions in which the final products are delivered to the environment. In the action rule of which the corresponding P-fact is the outcome, the existence of other P-facts may be stated as a necessary condition for establishing it. The same holds for the action rules concerning these 'component' P-facts. Put differently, a final product may be an assembly of parts, sub-parts etc. For example, in order to make John member of the library, Mary will ask John to pay the membership fee. This payment is considered to be a 'part' of the 'assembly' represented by the P-fact "membership #387 has started to exist".

3 The Transaction Pattern

In the previous section we have seen that an organization is a system of actors who perform two kinds of acts: production acts and coordination acts. But how are these acts related? Do they occur in particular structures or patterns or is any structure or pattern possible? At first sight, i.e. when surveying the enormous diversity of business processes in practice, the answer seems to be that there are at least many 'really' different processes. Fortunately, it appears that they can be considered as variants of one generic recurrent pattern of interaction between two actor roles. The pattern is called *transaction* and its universal regularity is called the *OER-paradigm*. The letters O, E and R refer to the three phases in which a transaction evolves: de *order* phase (O-phase for short), the *execution* phase (E-phase for short), and the *result* phase (R-phase for short). One of the two partaking actor roles is called the *initiator*, the other one the *executor* of the transaction. In the order phase, the initiator and the executor pursue to reach agreement about the production fact that the executor is going to produce for the initiator. In the execution phase, this production fact is actually brought about by the executor. In the result phase, the initiator and the executor pursue to reach agreement about the production fact that is actually produced (which might differ from the one that was originally requested). Only if this agreement is reached will the production fact become existing. The moment at which it starts to exist is the very moment of having reached agreement between the two actor roles (to be precise: between the two subjects that actually play the role of initiator and executor respectively). Every transaction is an instance of a particular transaction type; this transaction type corresponds with the type of the production fact that is the target/result of the transaction. For example, transactions regarding the P-fact type "membership M has started to exist" are of the same type. In section 2, the definition of actor role was provided. It can be made more specific now: an *actor role* is the 'amount' of authority that is necessary and sufficient to be the executor of exactly one transaction type. Being the executor of a transaction type includes being allowed to perform also all C-acts that are necessary to (eventually) perform the P-act.

3.1 The Basic Pattern and the Standard Pattern

The process of a transaction is a sequence of acts and resulting facts. Such a process can adequately and conveniently be modeled as a (colored) Petri-Net [2, 21]: the transitions (boxes in the Petri-Net Diagram) represent acts and the places (disks in the Petri-Net Diagram) represent facts. However, because of the special character of interaction processes, a special type of Petri-Net is introduced, called the CAP-Net. It is a further improvement of the net as presented in [5]. A distinctive feature of the CAP-Net, compared to the Petri-Net, is that every transition (act) has exactly one output place (fact). Figure 4 exhibits the CAP-Net of the basic pattern of a transaction. An open or white box represents a C-act type and an open or white disk represents a C-fact type. A gray box represents a P-act type and a gray diamond a P-fact type. The initial C-act is drawn with a bold line, as is every terminal C-fact. The gray colored frames, denoted by "initiator" and "executor" represent the *responsibility areas* of the two partaking actor roles. As an illustrating example we take the buying

of a loaf at the bakery's store. The customer plays the role of initiator, the baker plays the role of executor. The process starts with the request by the customer for delivering a loaf.

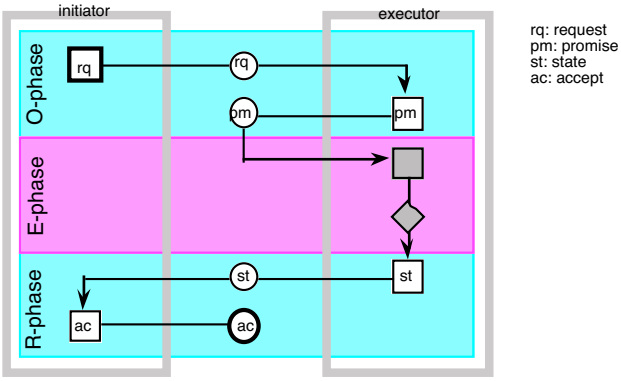

Fig. 4. CAP-Net of the basic pattern of a transaction

The result is the C-fact that the delivery is requested ("rq"). This C-fact is drawn between the two actor roles to show that it is a fact in their *intersubjective world* (cf. [14]). Both actors are allowed to know the fact. The C-fact "rq" is an agendum for the executor (the baker). As the outcome of dealing with the agendum, the baker promises to deliver the requested loaf, which brings the process in the state promised (the C-fact "pm"). This fact is an agendum for the baker. In dealing with it, the baker produces the P-fact. The P-act consists of the decision by the baker to sell the requested loaf to the customer. The P-fact is the transferred ownership of the loaf. The reason for coloring P-act types and P-fact types gray is to emphasize that they belong to the *subjective world* of the executor. They are principally not knowable to the initiator. Next, the baker states that the delivery of the loaf has been done, resulting in the (intersubjective) C-fact "st". This fact is an agendum for the customer. He responds to it by accepting the produced P-fact, which brings the process in the successful terminal state "ac". Figure 4 also shows the three phases of a transaction. The O-phase starts with the "request" act and ends with the state "promised". The E-phase starts with the "execution" act and ends with the state "executed". The R-phase starts with the "state" act and ends with the state "accepted".

At first sight, the whole process looks somewhat overdone. In particular the promise and the accept seem to be superfluous. They are not however; the basic pattern must always be passed through for establishing a new P-fact. A few comments are in place however. First, performing a coordination act does not necessarily mean that there is oral or written communication. Every (physical) act may count as a communicative act [24, 25]. For example, in the bakery's shop the "state" act is usually performed by putting the loaf on the counter in front of the customer. This act counts as performing the "state" act. Second, C-acts may be performed *tacitly*. Tacitly performing a C-act however is still performing that C-act! This becomes clear in case

of a breakdown [16,28]. In particular the promise and the accept are often performed tacitly. Further elaboration of this issue is beyond the scope of this paper.

The pattern exhibited in figure 4 is also called the success pattern, the course that is taken when the initiator and the executor keep *consenting* each other's acts. However, they may also *dissent*. There are two states where this may happen, namely the states "requested" and "stated". Instead of promising one may respond to a request by declining it, and instead of accepting one may respond to a statement by a reject. The reason for declining a request by the executor of a transaction or for rejecting a statement by the initiator, is in principle a mixture of the three *validity claims* of Habermas' theory [9, 14]. It brings the process in the intersubjective state "declined" or "rejected" respectively. These states are indicated by a double disk, meaning that they are *discussion* states. If a transaction ends up in a discussion state, the two actors must 'sit together', discuss the situation at hand and negotiate about how to get out of it. The basic pattern from figure 4, extended with the two dissent patterns is called the *standard* pattern. It is shown in figure 5.

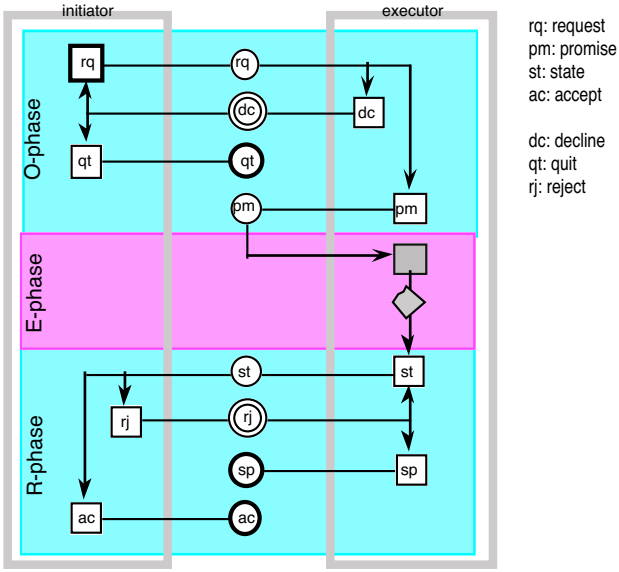

Fig. 5. CAP-Net of the standard pattern of a transaction

3.2 The Cancellation Patterns

In practice, it is quite common that either the initiator or the executor of a transaction wants to revoke an act (which may result in a partial or complete roll-back of the transaction). This is accommodated by the option to cancel any C-act in the basic pattern at any time. The CAP-Nets in the figure 6 through 9 exhibit all cancellation patterns. The acts and facts from the standard pattern are colored cyan in

order to distinguish them easily from the cancellation acts and facts. Every cancellation starts with a "cancel" act on which a conditional C-fact is put (represented by the dotted arrow). Only if the C-fact exists can the cancellation be performed. We take the bakery's shop example to explain the four cancellation patterns.

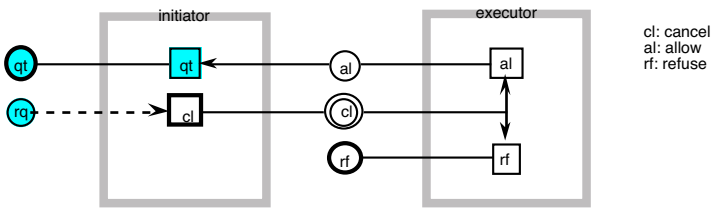

Fig. 6. CAP-Net of the cancellation of a request

Cancellation of the request occurs when the customer for some reason gets remorse about his initial request, e.g. because he sees in the shop a new kind of loaf that looks pleasant to him. The "cancel" act by the customer brings the process in the discussion state "cancelled". Normally, the baker will be inclined to please the customer and thus will allow the cancellation. From the state "allowed" that is reached then, the customer quits the transaction (and in this particular case immediately starts a new one). The baker may however very well refuse, e.g. if the transaction has already be completed and has taken place two days ago! The state "refused" is a terminal state for the cancellation, meaning that the C-fact "requested" remains to be the case.

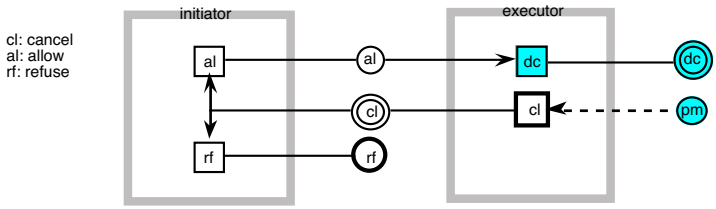

Fig. 7. CAP-Net of the cancellation of a promise

An example of canceling a promise is that the baker, after having promised, discovers that the last loaf of the kind the customer requested (which he had seen lying on the shelve) has meanwhile been sold by his assistant to another customer. If the customer allows the cancellation, the transaction process will be brought in the discussion state "declined" by the baker. From there they can negotiate about some other loaf. If the customer refuses, the C-fact "promised" remains to be the case. However, this doesn't help the customer much: he still will not get the loaf he wanted since there is none left. The transaction will forever stay in that state.

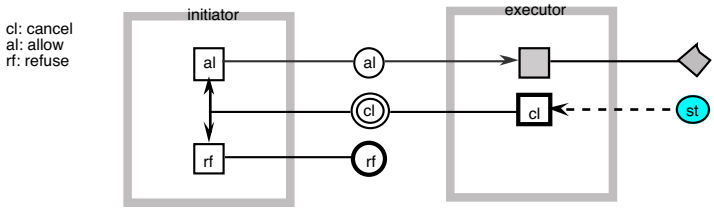

Fig. 8. CAP-Net of the cancellation of a statement

If the baker, after having put the loaf in front of the customer, discovers himself that the loaf is not quite okay, he may cancel his statement (e.g. in order to avoid the reject by the customer). After having explained in the discussion state "cancelled" why he did it, the customer will usually be inclined to allow the cancellation. The baker then redoes the execution act (decides to give the customer another loaf). But the customer may also refuse, which entails that he is willing to accept the loaf. The state "refused" is a terminal state for the cancellation, meaning that the C-fact "stated" remains to be the case.

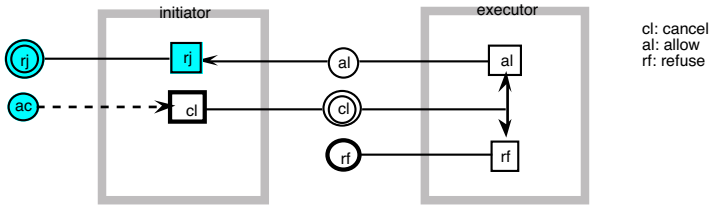

Fig. 9. CAP-Net of the cancellation of an acceptance

Suppose the whole transaction was (successfully) completed but, being outside the shop and having looked carefully at the loaf, the customer regrets his acceptance. He may then re-enter the shop and tell the baker that he wants to revoke his acceptance act. The reaction of the baker now probably depends on the relationship with the customer. If the baker considers the relationship important, he will allow the cancellation, after which the customer brings the transaction in the state "rejected". From there the most plausible act for the baker is to cancel his statement act (cf. figure 8 and the explanation given there). If the baker doesn't care much about the customer relationship, he may refuse, which means that the C-fact "accepted" remains to be the case.

For a complete understanding of the transaction pattern, the CAP-Nets of figures 6 through 9 must be superposed on the standard pattern of figure 5. The resulting *complete* pattern can rightly be called *universal* for the next reasons. First, the standard pattern has a firm theoretical legitimation in the social action theory of Habermas [9, 14]. Second, the four cancellation patterns (superposed on the standard pattern) are necessary and sufficient to allow for all possible rollback requirements. Every instance of any transaction type is some course through this complete pattern.

4 Business Processes

The transaction concept, as introduced in the previous section, is the *molecular* component of business processes, the coordination and production acts and facts being the *atoms* [13]. We will demonstrate in this section how business processes can be modeled as compositions of transactions. As the illustrating example, the case of a pizzeria is taken from [13]. The starting point in modeling is the identification of the *independent* transactions, i.e. the transactions in which final products are delivered to the environment. By applying the P-CAP (Core Activity Principle) to the pizzeria case, it appears that the final product (T1: delivering order) is an 'assembly' of three 'parts': dealing with the baking of the pizza's of an order, the transporting of these pizza's to the customer's address, and the payment of the order. They are listed in figure 10 as T2, T4 and T5. T2 is itself an 'assembly' of a (only at run time known) number of T3's. The variable 'O' denotes an instance of a customer order; the variable 'P' denotes a particular pizza kind. For example, if an order consists of two pizza's Vesuvio and one Capriccioso, there are three different pizza instances in the order.

transaction	resulting P-fact
T1 delivering_order	F1 *order O has been delivered*
T2 baking_order	F2 *the pizzas of order O are baked*
T3 baking_pizza	F3 *a pizza of kind P is baked*
T4 transporting_order	F4 *the pizzas of order O are transported*
T5 paying_order	F5 *order O has been paid*

Fig. 10. Interaction Table of the pizzeria case

It is convenient to have a simple symbol for the complete pattern of a transaction in order to draw very concise (but at the same time very precise!) models of business processes. The transaction symbol used in DEMO consists of a diamond (the production symbol) 'embedded in' a disk (the coordination symbol). Figure 11 shows the so-called *Interaction Diagram* of the pizzeria case, in which the transaction symbol occurs, next to a box for actor roles. By convention, the executor of a transaction type gets the same number as the transaction type. So, the executor of T1 in the pizzeria case is A1. The initiator of T1, who apparently is an actor in the environment of the pizzeria, is 'hidden' in the system actor S1. From the product structure of the final product, it follows that A1 is the initiator of transaction types T2, T4 and T5. The executors of these transaction types are respectively numbered A2, A4 and A5 (who is 'hidden' in S1). Actor role A2 is the initiator of transactions T3, of which A3 is the executor. A transaction symbol is connected through straight lines with its initiator and executor. The small black box on the edge of an actor box at the junction with a transaction link indicates the executor. Because the initiator of T1 is an actor in the environment of the organization (hidden in S1), transaction T1 is drawn on the *system boundary*. The same holds for T5, of which the executor is an actor role in the environment (also hidden in S1). The actor roles A1, A2, A3 and A4 are inside the system boundary, meaning that they belong to the responsibility of the pizzeria.

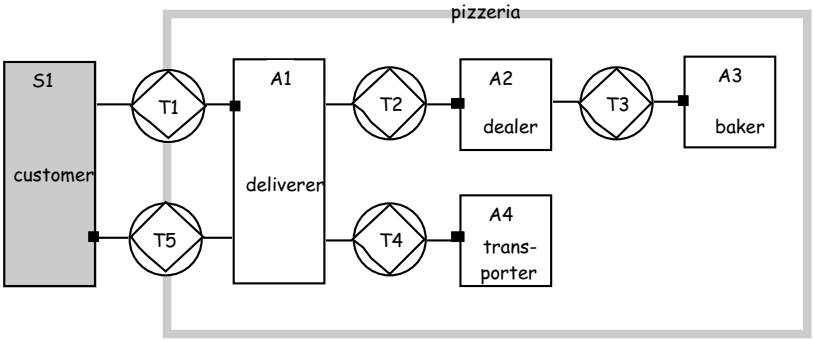

Fig. 11. Interaction Diagram of the pizzeria case

The transaction symbol also occurs in the *Process Phase Diagram* (figure 12). It exhibits the causal and conditional relationships between transactions. There is a proportional time axis from top to bottom. The symbols of T1 and T2 are stretched in order to accommodate the placing of the other transactions. T1 is initiated externally. T2, T4 and T5 are initiated from the E-phase of T1, transaction T3 is initiated from the E-phase of T2. This is denoted by the arrows (*causal links*) to the 'top' of the respective symbols. The actual start of T4 has to wait for the completion of T2. Likewise, T5 has to wait for the completion of T4, the execution of T2 waits for the completion of all T3's within the same order, and the execution of T1 waits for the completion of T5. Waiting is denoted by the dotted arrows (*conditional links*).

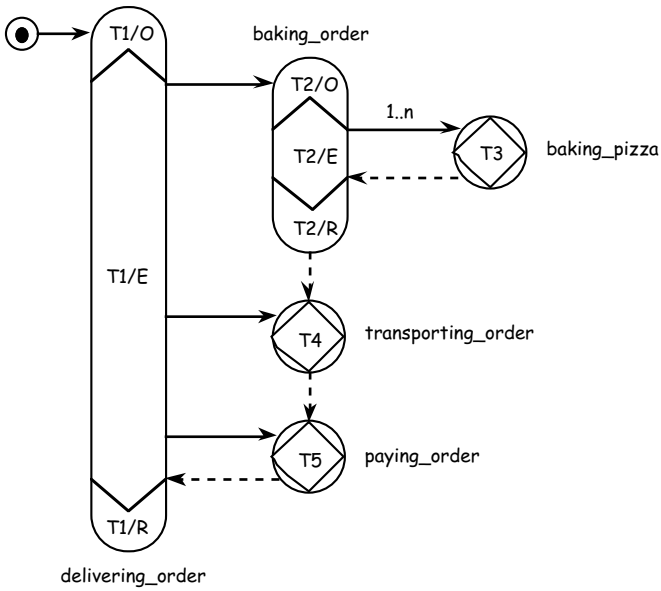

Fig. 12. Process Phase Diagram of the pizzeria case

The Process Phase Diagram suggests a convenient and rigorous definition of a business process: a *business process* is an independent transaction type together with all its dependent transaction types, where dependent means: causally linked.

Figure 13 exhibits the *Process Step Diagram* of the business process of figure 12. A Process Step Diagram is a compact CAP-Net: instead of drawing separately a box for an act X and a disk for the resulting fact X, the disk is drawn inside the box. X now stands both for the intention of the act (e.g. request) and for the intention of the resulting fact (e.g. requested). For simplicity, only the basic pattern of the transactions is drawn. The arrows have the same meaning as in figure 12.

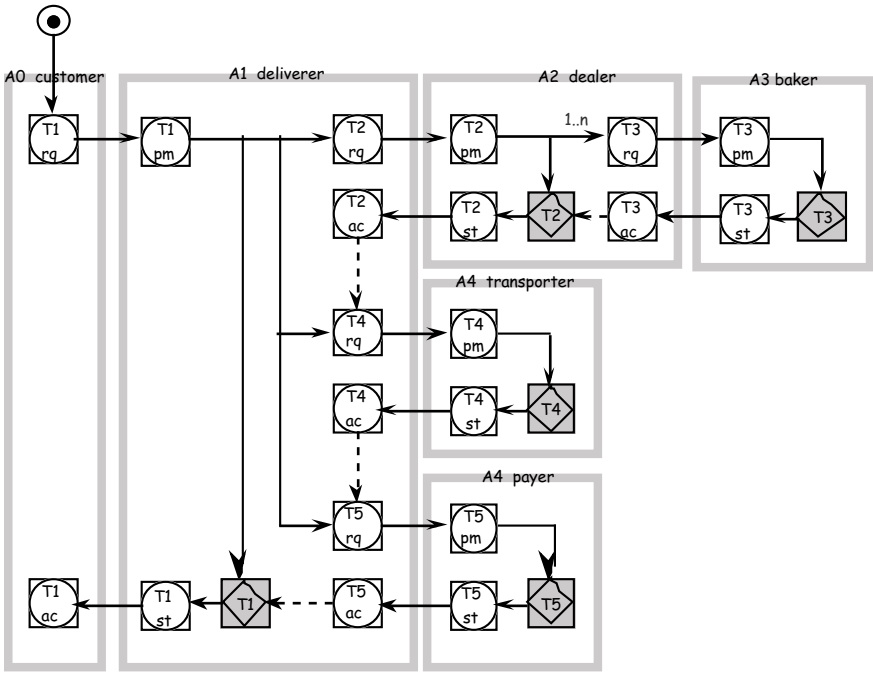

Fig. 13. Process Step Diagram of the pizzeria case

5 Discussion and Conclusions

Most publications concerning the application of generic patterns in modeling processes are not so much about business processes but rather about workflows. Although of course 'a word is just a word', we like to plea strongly for maintaining a strict distinction between the implementation and technology independent notion of *business process* (like the one presented in this paper) and an implementation related notion. The last one is commonly referred to by the term '*workflow*' (cf. the definition by the Workflow Management Coalition). A few publications can be found regarding

workflow patterns, e.g. [1, 8]. The common interest in these papers is to specify workflows for the purpose of 'programming' workflow management systems. Because of that aim, emphasis is put on formality and correctness. Consequently, easily verifiable modeling techniques, like Petri-Nets, are applied. This interest is quite different from the interest in the design and analysis of business processes, as exposed in this paper. Nevertheless, an attempt is made to evaluate the transaction pattern with the patterns discussed in [3]. The outcome is that they can all be accomodated (Note. Lack of space prohibits to give a full account of the evaluation).

The first conclusion to be drawn is that a considerable effort has to be made yet to bridge the gap between the abstracted transaction pattern and the concrete workflow patterns that are used in specifying workflows for the implementation of workflow systems. The feasibility of deriving workflows from DEMO business processes however, has been demonstrated already in [20], which reports about the automatic translation of Process Step Diagrams into Petri-Nets. A second conclusion is that the presented generic transaction pattern is a very valuable help for the modeling and understanding of business processes, abstracted from their implementation. In several aspects (like the Core Activity Principle) it differs from similar approaches [18, 19, 26, 27]

Over sixty *practical projects* have been coducted in the past ten years. The common experience is that DEMO offers several advantages: comprehenisibility, modularity, completeness, integrality, consistency, objectivity, and a substantial reduction of duration and costs. There are several *research projects* running in which the DEMO transaction concept plays a crucial role. One of them is the derivation of use cases from business process models. Another one is the modeling of health care processes using the transaction concept. These processes are among the most complicated business processes, where all of the twenty patterns listed in [3] occur time and again. A third one is the development of a generic information systems component that supports the complete pattern of a transaction type and keeps track of the progress of all of its instances, running sequentially or in parallel. Among the recently proposed projects is one that concerns the development of transaction-based architectures for the alignment of business and ICT-applications.

References

1. Aalst, W.M.P., The Application of Petri Nets to Workflow Management, *The Journal of Circuits, Systems and Computers*, 1998
2. Aalst, W.M.P. van der, Hee, K.M. van, *Workflow Management: Models, Methods and Systems*, MIT Press, MA, 2001
3. Aalst, W.M.P. van der, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Baros, Workflow Patterns, 2002 (to appear in *Distributed and Parallel Databases*)
4. Austin,J.L., *How to do things with words*, Harvard University Press, Cambridge MA, 1962
5. Barjis, J., Dietz, J.L.G., Business Process Modeling and Analysis Using GERT Networks, *Proc. of the 1st International Conference on Enterprise Information Systems*, vol II, pp 748-756, ISBN 972-98050-0-8.

6. Bunge, M.A., *Treatise on Basic Philosophy*, vol.3, *The Furniture of the World*, D. Reidel Publishing Company, Dordrecht, The Netherlands, 1977
7. Bunge, M.A., *Treatise on Basic Philosophy*, vol.4, *A World of Systems*, D. Reidel Publishing Company, Dordrecht, The Netherlands, 1979
8. Christensen, S., Petrucci, L., Towards a Modular Analysis of Coloured Petri Nets, in: Jensen, K., *Application and Theory of Petri Nets*, LNCS 616, Springer-Verlag, 1992.
9. Dietz, J.L.G., G.A.M. Widdershoven, Speech Acts or Communicative Action? *Proc. 2nd European Conf. on CSCW*, Kluwer Academic Publishers, Boston, 1991
10. Dietz, J.L.G., G. Goldkuhl, M. Lind, V.E. van Reijswoud, The Communicative Action Paradigm for Business Modeling, in: Goldkuhl, G., Lind, M., Seigerroth, U. (eds.), *Proc. of the 3rd LAP workshop*, Jönköping Int. Business School, 1998
11. Dietz, J.L.G., Understanding and Modeling Business Processes with DEMO, in: *Proceedings of the ER Conceptual Modeling Conference*, Paris, 1999
12. Dietz, J.L.G., DEMO: Towards a discipline of organisation engineering, *European Journal of Operational Research*, nr. 128, 2001, pp 351–363
13. Dietz, J.L.G., The Atoms, Molecules and Fibers of Organizations, 2003 (to appear in *Data and Knowledge Engineering*)
14. Habermas, J., *Theorie des Kommunikatives Handelns*, Erster Band, Suhrkamp Verlag, Frankfurt am Main, 1981
15. Hee, K.M. van, Houben, G-J., Dietz, J.L.G., Modelling of discrete dynamic systems; framework and examples, *Information Systems*, vol 14, 1989.
16. Heidegger, M., *Sein und Zeit*, Neomarius Verlag, Tübingen, Germany, 1927
17. Knolmayer, G., R. Endl, M. Pfahrer, Modeling Processes and Workflows by Business Rules, in: Aalst, W. van der, J. Desel, A, Oberweis (Eds.), *Business Process Management*, Lecture Notes in Computer Science 1806, Springer-Verlag, 2000
18. Lind, M, Goldkuhl, G., Generic Layered Patterns for Business Modelling, in: Schoop, M., Taylor, J. (eds.), *Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling*, RWTH Aachen, 2001
19. Medina-Mora, R., T. Winograd, R. Flores, F. Flores, The Action Workflow Approach to Workflow Management Technology. In: J. Turner, R. Kraut (Eds.), *Proceedings of the 4th Conference on Computer Supported Cooperative Work*. ACM, New York
20. Oren, E, *Het vertalen van Designer Modellen naar ExSpect*, IS research report, Delft University of Technology, 2002 (in Dutch).
21. Peterson, J.L., 1981. *Petri net theory and the modeling of systems*. Prentice-Hall, Inc., Englewood Cliffs, NJ.
22. Porter, M.E., 1985. *Competitive Advantage, Creating and sustaining superior performance*. The Free Press, New York.
23. Reijswoud, V.E. van, J.B.F. Mulder, J.L.G. Dietz, Speech Act Based Business Process and Information Modeling with DEMO, *Information Systems Journal*, 1999
24. Searle, J.R., *Speech Acts, an Essay in the Philosophy of Language*, Cambridge University Press, Cambridge MA, 1969
25. Searle, J.R., *The Construction of Social Reality*, Allen Lane, The Penguin Press, London, 1995
26. Schäl, T., 1996. *Workflow Management Systems for Process Organisations*. Lecture Notes in Computer Science 1096, Springer, Berlin.
27. Stamper, R., Liu, K., Hafkamp, M. & Ades, Y. (2000) "Understanding the Roles of Signs and Norms in Organizations", *Journal of Behavior and Information Technology*.
28. Winograd, T, F. Flores, 1986. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, Norwood NJ.

Personal Schedules for Workflow Systems

Johann Eder, Horst Pichler, Wolfgang Gruber, and Michael Ninaus

University of Klagenfurt
Institute for Informatics-Systems
{eder,horst,wolfgang}@isys.uni-klu.ac.at
mninaus@edu.uni-klu.ac.at

Abstract. Personal schedules allow workflow participants to improve their performance of activity executions. Participants are no longer surprised by the entries in their work-lists but receive advance information about (potential) future activity assignments, allowing better possibilities for work-planning. The personal schedule system is based on a probabilistic workflow time management system using duration histograms. A personal schedule collects future activity assignments together with their probability and their timing requirements and allows to analyze the workload of a participant and to support the scheduling of activities with the goal of reduced turn-around times and reduced number of violations of temporal constraints.

Keywords: workflow system, time plans, temporal constraints, scheduling, personal scheduling

1 Introduction

In the execution of workflows, workflow participants are typically "surprised" by the activities they should perform, surprised in the sense that they find these activities in their to-do-lists when these activities are ready, i.e. all preceding activities are finished. Information about upcoming activities would be much earlier available in the workflow system. For an example, when the first activity of a sequence is ready, the succeeding activities will be ready soon. Current workflow systems do not make use of this information and do not forward this information to the participants depriving them of the possibility of planning their work ahead. For administrative processes, in particular in settings where workflow participants have dispositive competencies and have to manage their schedules this strategy leads to suboptimal results. The main shortcomings are the following:

- longer retention period of activities in work-lists before they are taken up
- longer turnaround time
- no workload balancing
- considerable number of deadline violations

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 216–231, 2003.

Reasons for this situation might be that workflow systems typically do not compute schedules due to the partial knowledge they have about the execution of their processes, the impossibility to know the actual flow at decision points at process instantiation time and typical variance in the execution of individual tasks. Nevertheless, several research proposals and prototypical implementations (e.g. [4,6,2,8]) propose improvements. The main idea of these efforts is to make best use of the available information. The approach presented in this paper follows these directions. We aim at improving the planning situation of workflow participants by making them information about their future workload available early and provide them with means for digesting and using this information.

For an improvement of the sketched planning situation we envision the following scenario: In current workflow systems every participant owns a work-list which provides information about every activity that has been assigned to him. These activities may arrive from several workflow systems and stem from several instances of one specific workflow. We enhance the idea of the work-list to a personal schedule which contains not only the "ready" activities but also those which might become ready in near future. E.g. when a workflow-instance starts and the first activity is assigned to a participant the actors of all succeeding activities receive information about tasks that may be assigned to them soon, together with temporal characteristics, constraints and the probability of the assignment.

In practice we find that people are involved in workflows managed by different workflow systems and have additional responsibility not supported by any workflow system. So general central scheduling approaches like they have been developed and are used in production management and ERP systems cannot not be used in such environments. Personal schedules are primarily intended as a support structure for individuals organizing the execution of their workload in time striving for improved performance.

Our personal schedule approach is based on a probabilistic time management system [3] which uses duration histograms to express the uncertainty of workflow time plans stemming from variations in activity execution durations and from the unpredictability of control decisions.

The rest of the paper is organized as follows: In section 2 we introduce our workflow model and all basic definitions used in the following, and we summarize our probabilistic time management concept. In section 3 we show how to compute time plans for workflows. In section 4 we present the concept of personal schedules, and in section 5 the algorithms for the computation of personal schedules and their manipulation is covered. In section 6 we discuss some applications and draw some conclusions.

2 Workflow Model and Time Histograms

2.1 Basic Definitions and Assumptions

We define a rather generic structured workflow model that we use in the rest of this paper and introduce the *time plan* on base of a *probabilistic timed graph*

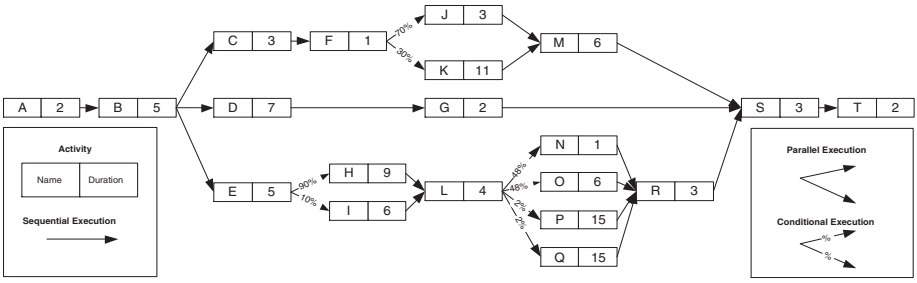

Fig. 1. Example workflow process schema

as a structure for representing probabilistic information about the duration of activities and processes.

Essentially, a workflow is a collection of *activities,* and *dependencies* between activities. Activities correspond to individual steps in a business process. Dependencies determine the execution sequence of activities and the data flow between them. Activities can be executed sequentially, in parallel (and-splits) or conditional (or-splits). Consequently, a structured workflow can be represented by a directed acyclic graph, where nodes correspond to activities and edges correspond to dependencies between activities. Additionally the model contains the expected duration for each activity and statistically weighted values for each conditional branch, defined by administrator estimations or average values from past executions.

Figure 1 shows an example workflow schema. The 3 routes after B (*and-split*) will be processed concurrently. The workflow continues with S (*and-join*) not until M, G and R are finished. An example for a conditional execution is E. In this case only one specific path after E (*or-split*) will be chosen, which results in 2 different possible execution-routes from E to L (*or-join*) and 8 possible routes between E and T. H will be executed after E in 9 out of 10 cases and O will be executed after L in 4.8 out of 10 cases. Assuming that each branching decision is an independent event the probability that a workflow-instance will execute the path from E via H and O to T can be calculated as $0.9 \cdot 0.48 = 0.432$.

2.2 Time Histograms

As stated above activities can have multiple start-times and end-times due to conditional branches depending on the execution path. To represent probabilistic values of possible start-times and end-times (complex) activities we enhance the idea of *duration histograms* [3] to *start time histograms* and *end time histograms* which will subsequently be merged into the *probabilistic timed graph.*

To represent these non-scalar values we use time-probability tuples. E.g., the path from A to E is sequential and unambiguously determined, therefore E holds a set with only one start-time tuple $(1.0, 7)$ where the second value specifies

the start-time and the first value specifies the according execution-probability. The start-time is calculated by adding the duration of all predecessor-activities. The activity L can be reached on two routes (via H and via I) each with its own execution probability, thus L holds a set of two different start-time tuples $\{(0.1, 18), (0.9, 21)\}$. Note that the probabilities always sum up to 1.0.

In contrast, according to ePERT and similar approaches [10,4,6], end-times are interpreted as upper bounds for the execution of an activity. For the calculation we have to change our point of view on the workflow-process and start from the last activity T, initialized with a given deadline δ, which can be provided by an administrator or calculated from start-times. Furthermore we treat splits as joins and vice-versa. E.g., the reverse-path from T to R is sequential and unambiguously determined, therefore R holds a set with only one end-time tuple $(1.0, 43)$. The second value specifies the end-time, which is calculated by subtracting the duration of all successor-activities from an assumed deadline $\delta = 48$. Activity L can be reached on four routes (via N, O, P and Q), thus it would have four end-times tuples $(0.02, 25), (0.02, 25), (0.48, 34), (0.48, 39)$.

Since our approach is set based and we do not need any knowledge about coherences between tuples and paths anyway, it is possible and necessary to aggregate tuples with equal time-information by adding their probability. Which results in a set with three different tuples $\{(0.04, 25), (0.48, 34), (0.48, 39)\}$ for activity L.

These sets can be represented as histograms which are formally defined as follows:

Definition 1 (Time Histogram) *A time histogram H is a binary relation with n rows (p, t) with probability p and time-information t.*

A time histogram H is valid, if $\sum_{i=1}^{n} p_i = 1$ for $(p_i, t_i) \in H$.

An extended time histogram T is a relation of n rows (p_i, c_i, t_i), (probability p, cumulated probability c, and time-information t), with $\sum_{i=1}^{n} p_i = 1$, and $c_i = \sum_{t_j \leq t_i} p_j$ for $1 \leq i \leq n$.

A cumulated time histogram is the projection of an extended time histogram on the cumulated probabilities and the time information.

The various start-times of an activity are represented in a *start time histogram* or *S-histogram*, where the time t is represented by the start-time-value s. The upper-bounds are stored in an *end time histogram* or *E-histogram*, where the time t is represented by the end-time-value e. The *probabilistic timed graph* (see figure 2) is generated by calculating the S-histograms and E-histograms of all activities.

2.3 Calculation of the Probabilistic Timed Graph

The calculation of the graph starts with the initialization of the start activity's S-histogram with $(1.0, 0)$. To determine the S-histograms for all remaining activities the forward-calculations specified below have to be applied depending on the corresponding control structures (sequence, conditional, parallel).

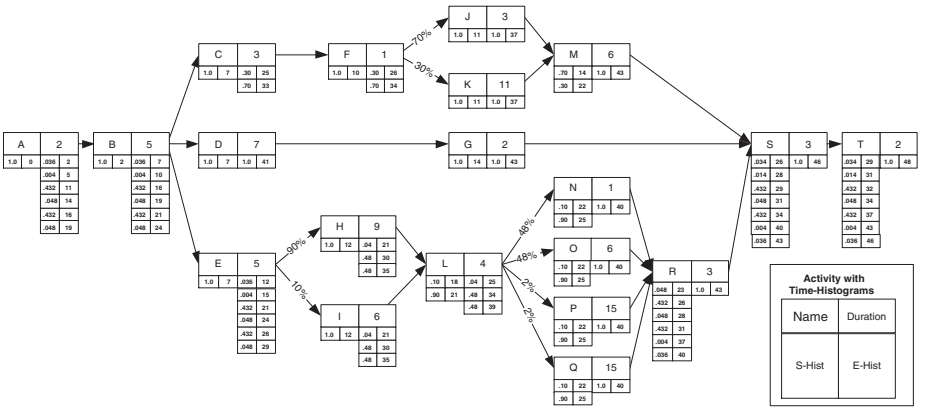

Fig. 2. Probabilistic timed graph with S-histograms and E-histograms

Definition 2 (Sequential Execution) *For two sequential activities B and C we define the S-histogram of the successor activity C on basis of the predecessors S-histogram S_B and its duration d_B as*

$$S_C = S_B + d_B = \{(p, s + d_B)|(p, s) \in B\}$$

For the successors S-histogram the predecessors duration d is added to each start-time s of the predecessor. Examples for the calculation of sequences are the activities B, D, E, O and T. Note that this calculation rule has to be applied for all successor activities of sequences as well as all activities located after split-nodes. The calculation of join-nodes vary dependent on their type.

Definition 3 (Conditional Execution) *Let B_i, with duration d_i and S-histogram S_{B_i}, be conditionally executed predecessors for or-join activity C and q_i the corresponding branching probabilities, then the S-histogram S_C is defined as*

$$S'_C = \{(B_i, p * q_i, s + d_i)|(p, s) \in S_{B_i}\}$$
$$S_C = \{(\sum y, z))|(x, y, z) \in S'_C\}$$

Each tuple (p, s) of every predecessors S-histogram is aggregated into the successors S-histogram by adding the predecessors duration d_i to the start time s and weighting the corresponding starting-probability p with the predecessors branching factor q_i. The set S'_C is an intermediate step to distinguish tuples with equal probabilities which are finally aggregated in the calculation of S_C (as described in the example in section 2.2). Note that the sum of all probabilities of the resulting histogram is always 1, and thus it is a valid time histogram. Examples are the activities L, M, and R in Figure 2.

After and-splits all succeeding routes will be processed concurrently. To calculate the S-histogram for activities located after and-joins we define the following operation.

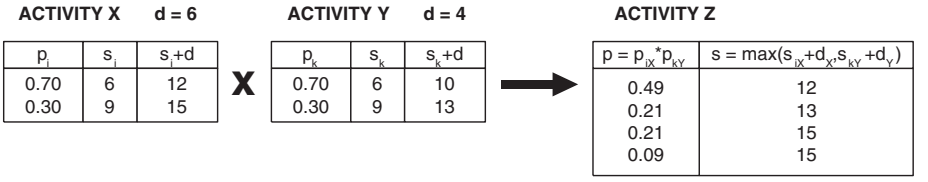

Fig. 3. Parallel execution of X and Y joining in Z

Definition 4 (Parallel Execution) *Let B_1 and B_2 be predecessors with S-histograms $S_{B_{1,2}}$ and durations $d_{B_{1,2}}$ which are executed in parallel before and-join activity C, then the successors S-histogram S_C is defined as*

$$S'_C = \{(p_{B_1}, p_{B_2}, p_{B_1} * p_{B_2}, S_{B_1}, S_{B_2}, max((s_{B_1} + d_{B_1}), (s_{B_2} + d_{B_2}) \mid (p_{B_1}, s_{B_1}) \in S_{B_1}, (p_{B_2}, s_{B_2}) \in S_{B_2}\}$$
$$S_C = \{(\sum w, z) | (u, v, w, x, y, z) \in S'_C\}$$

Again S'_C is used as intermediate step for probability aggregation. This definition can be extended to any number of predecessor activities due to the operations associativity. An example result for three predecessors can be found in figure 4 (S-histogram of and-join activity S). An example calculation for two predecessors is illustrated in figure 3 where 2 parallel activities X and Y with $S_X = \{(0.7, 6), (0.3, 9)\}$, $S_Y = \{(0.7, 6), (0.3, 9)\}$, $d_X = 6$ and $d_Y = 4$ join in activity Z. Note that we calculate the successor start-times from each predecessors start-time and duration $s_{B_i} + d_{B_i}$. Assuming that the concurrent activities are completely independent from each other, all possible end-time combinations have to be calculated whereas the greater end-time and the product of the probabilities are chosen for the resulting S-histogram.

Before starting the backward-calculation it is necessary to initialize the E-histogram of the end-activity with $(1.0, \delta)$. δ is the workflows deadline, which can be chosen freely or calculated as structural deadline $\delta = max(s|(p, s) \in S_Z$ where Z denotes the workflows last activity. As stated above we have to reverse our point of view on the workflow, that means the calculation starts from the last activity, splits are treated as joins and vice-versa. The calculation-algorithms for S-histograms (see definitions in section 2.3) must be modified as follows: Calculate the E-histograms of activities on base of their successors (instead of predecessors). For sequences and conditionals subtract the successor durations d from their end-times e (instead of adding to s). For parallel execution calculate the predecessors end-times from the successor end-times and durations as $e - d$ (instead of their start-times $s + d$) and use the min-operator instead of the max-operator.

2.4 Additional Issues on Time Histograms

For further issues in computations with time histograms we refer to [3]. Such issues are representation and calculation of iterations, compression of time histograms, checking the satisfiability of temporal constraints, and run-time assessments of the temporal situation of workflow execution.

3 Time Plans

The probabilistic timed graph is the basis for the calculation of time plans which contain probabilistic information about execution intervals of activities.

3.1 X-Values

There is still one essential piece of information, which the probabilistic timed graph does not provide for time plans: The probability that an activity will not be assigned and executed at all. E.g. in figure 2 the probability that the path via activity H will be chosen is 90%. Based on the fact that conditional branching-decisions are unknown in advance, the "uncertainty-factor" that any other path will be chosen and the activity will not be assigned is 10%. For L it is 0% because this activity will be assigned and executed in any case.

That knowledge may be of no interest in an overall process view, but for a participant who is supposed to execute an activity it is crucial. This information is stored in the *X-value* of each activity in the workflow.

Definition 5 (X-Value) *Let x_A be the X-value of an activity A, such that x_A specifies the probability of not executing A.*

The calculation of the X-value can be linked to the forward-calculation of the probabilistic timed graph and can be determined for every activity as follows (see also figure 4):

- Initialization of first activity with $X_{FirstActiviy} = 0$.
- For two sequential activities B and C, the X-value of the successor activity C is $X_C = X_B$.
- For an activity C which succeeds an or-split activity B with a branching probability q_C the X-value is $X_C = 1 - ((1 - X_B) * q_C)$.
- For an or-join activity C which succeeds multiple activities B_i, where $2 \leq i \leq n$, the X-value is $X_C = 1 - \sum(X_{B_i})$.
- For an activity C which succeeds an and-split activity B the X-value is $X_C = X_B$.
- For an and-join activity C which succeeds multiple activities B_i, where $2 \leq i \leq n$, the X-value is defined as $X_C = X_{B_1} = X_{B_2} = ... = X_{B_n})$.

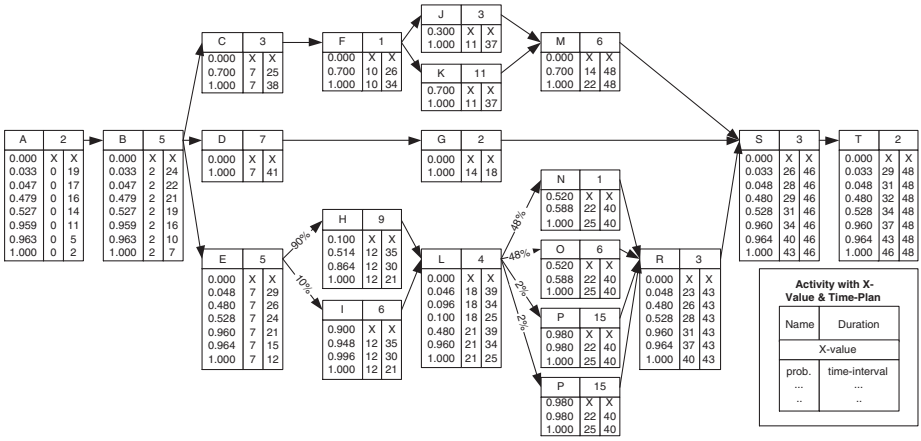

Fig. 4. Workflow with calculated time plans

3.2 Calculation of Time Plans

A time plan that holds all possible execution-intervals for an activity is defined as follows:

Definition 6 (Time Plan) *The time plan T_A on an activity A is a set of tuples (p, s, e) which is calculated on basis of A's cumulated S-histogram S_A, its cumulated E-histogram E_A and its X-value x_A where*

$$T_A = \{(c_S * c_E * (1 - x_A) + x_A, s, e) | (c_S, s) \in S_A \wedge (c_E, e) \in E_A\}$$

Note that we used cumulated histograms which can be determined as stated in definition 1. Basically we create the cartesian product of the cumulated S-histogram and the cumulated E-histogram weighted by the activity's X-value, which takes the potential non-assignment of the activity into account. One tuple (p, s, e) defines the probability p that no time-constrained is violated (deadline) when the activity starts and ends in the time-interval $[s..e]$. Figure 4 shows our example workflow with calculated time-plans for all activities. In this example each activity holds a tuple with probability $p = 1.0$, which means that an execution of the particular activity without constraint-violation is possible in the according time-interval.

Definition 7 (Safe Interval) *A time plan T_A of an activity A is safe*

$$iff \ \exists (p, s, e) \in T_A \ with \ p = 1.0$$

and (s,e) is called the ***safe interval*** *of A.*

Taking a closer look on the time plan of activity H in figure 4, there are some conclusions that can be drawn: (a) H will not start before 12, (b) if H starts

and ends in the safe interval [12..21] an execution without violating any time-constraints can be guaranteed, (c) in time interval [12..30] there exists an 86.4%-chance that an execution without violating any time-constraints is possible, (d) in time interval [12..35] there exists an 51.4%-chance that an execution without violating any time-constraints is possible and (e) according to the X-value there is 10% chance that H will not be executed at all.

4 Definition of Personal Schedules

Based on time plans one can calculate future workloads for participants in terms of personal schedules, which are not comparable with machine scheduling plans, because a participant still has the freedom of choice whether he executes a certain activity or not and when he executes it. They are intended for the support of predictive personal scheduling issues by providing knowledge about upcoming activities, possible future bottle-necks and the detection of upcoming violations of time constraints as early as possible.

We assume that each workflow participant is responsible for executing activities from different workflows and different workflow instances. Therefore a number of *instance activities* including time plans and X-values are provided for a participant from the workflow-system(s). Note that these activities are possible future-tasks of the participant, but may not be assigned at all due to branching decisions that lead to different execution-routes in the workflow(s). Such future assignments of activities are represented as *personal schedules* formally defined as follows:

Definition 8 (Personal Schedule) *A personal schedule PS_ρ of participant ρ is defined as a set of tuples (v, p, s, e) with $(p, s, e) \in T_v$ and:*

v ... *activity that participant ρ must work off*
T_v ... *time plan of activity v*
p ... *probability to meet execution of v in (s, e)*
s ... *planned start time of activity v*
e ... *planned end time of activity v*
($e = s + d$, d ... execution time of activity v)

For each activity the personal schedule PS contains a planned execution interval $[s..e]$ and the corresponding probability p to meet execution within this interval from the time plan T_v. A personal schedule is not an optimized plan but only one possible execution order used for workload calculations. Since we are working on administrative workflows, the real execution order is always up to the participant himself. Nonetheless we envision a situation where personal schedules support participants in choosing a most efficient execution order in the context of all integrated workflows.

We assume that instance activities of one participant can not be executed in parallel. Therefore the planned execution intervals of a personal schedule must not overlap:

activity instance A

A	6	
0.00	X	X
0.90	0	14
1.00	4	14

activity instance B

B	10	
0.00	X	X
0.60	8	22
1.00	8	20

activity instance C

C	12	
0.00	X	X
1.00	18	30

Fig. 5. Example: future activities including time plans for one participant

Definition 9 (Non-overlapping) *A personal schedule PS is called non overlapping iff* $\forall(v_1, p_1, s_1, e_1) \in PS$ *and* $\forall(v_2, p_2, s_2, e_2) \in PS$ *where* $(v_1, p_1, s_1, e_1) \neq (v_2, p_2, s_2, e_2) \rightarrow \neg((s_1 < s_2 < e_1) \vee (s_1 < e_2 < e_1))$.

As personal schedules predict future workload using probabilities, we are faced with uncertainties and estimations. In contrast to production planning systems, the purpose here, is not to find the optimal plan. Moreover in this context this would lead to a schedule algorithm which is known to be np-hard [1]. We therefore reduce complexity by using *earliest deadline first strategy*. That means we are planning activities consecutively starting with the activity having the earliest deadline and so on.

5 Calculation of Personal Schedules for Workflows

Having time plans for all future activities of a participant enables one to make some predictions about the work-list behavior of this participant in the near future. We calculate personal schedules as a simulation of the activities that will be added to the work-list of a participant. This way potential overload can be detected and the probability for this overload occurring can be determined as well. In this section we show how to calculate personal schedules and how to interpret them.

To illustrate the steps of the algorithm we introduce the following example: Figure 5 shows some activities that one participant is supposed to execute in the near future and for which the personal schedule calculation will be calculated.

For each activity the time plan contains a safe interval, in which the execution should be scheduled if somehow possible. The safe interval of an activity is that period of the time plan, for which the probability is $p = 1.0$ (as described in definition 7 above). Therefore the safe intervals for the activities of the example are (see figure 5 and figure 6):

- safe interval of $A = [4..14]$,
- safe interval of $B = [8..20]$,
- safe interval of $C = [18..30]$.

The objective of the algorithm is to find an execution period for all activities without overlaps, which should furthermore lie within their safe intervals. An overlap would mean to risk temporal restriction violations due to an overload

on the participant. And execution outside the safe interval can lead to temporal constraint violations in the associated workflow. Nevertheless, even if no non-overlapping execution period in the safe interval can be determined for an activity, the probability, with which the entire personal schedule can be held will be computed. This value can be important for decisions about continuation or interruption of activities and workflows.

This pseudo-code describes the calculation of a personal schedule in detail followed by some explanatory remarks based on the example.

```
Input V      set of activities including time plans
      type   normal, min or max personal schedule
      period for min or max personal schedule
Out   PS     personal schedule

FUNCTION calculatePersonalSchedule(V, type, period)
BEGIN
   V.sort() //sort activities by their deadlines
   PS := ∅
   noPlanFound = FALSE
   FOR every (v ∈ V) AND WHILE (NOT noPlanFound)
      bestPlan := ∅
      bestPeriodFound = FALSE
      v.timePlan.sort() //sort time plan entries descending by probabilities
      FOR every (period ∈ v.timePlan) AND WHILE (NOT bestPeriodFound)
         IF (period.startTime = 'x') THEN
            //X-value, activity is planned to be not executed
            bestPlan.weight(period.getProbability())
            bestPeriodFound = TRUE
         ELSE
            //find non-overlapping time interval in period
            newPeriod := getFreePeriod(PS, period, v.executionTime)
            newPeriod.probability = period.getProbability()
            //if period is safe
            IF (newPeriod.getProbability() = 1.0) THEN
               check constraints for minimized or maximized personal schedule
               bestPlan.add(a, newPeriod)
               bestPeriodFound = TRUE
            //if period is overlapping with other intervals
            ELSE IF (newPeriod.getProbability() = 0.0) THEN
               newPeriod.endTime := period.endTime
               newPeriod.startTime := newPeriod.endTime - v.executionTime
               //try to shift conflicting activities
               newPlan := freePeriod(bestPlan, newPeriod)
               IF (newPlan ≠ ∅) THEN
                  bestPlan := newPlan
                  bestPlan.add(a, freePeriod)
               END-IF
            //if period is not safe but non-overlapping
            ELSE
               newPlan = PS
               newPlan.add(a, newPeriod)
               IF (newPlan.getProbability() > bestPlan.getProbability()) THEN
                  bestPlan := newPlan
               ELSE
                  bestPeriodFound = TRUE
               END-IF
            END-IF
         END-IF
      END-FOR
      IF (bestPeriodFound = FALSE) THEN
         noPlanFound = TRUE
      ELSE
         PS := bestPlan
```

```
      END-IF
   END-FOR
   RETURN PS
END

function getFreePeriod()
   Returns a time interval of the specified length that lies within the
   specified period, if such an interval exists without overlapping with
   other intervals of the specified personal schedule.

function freePeriod()
   Returns a new personal schedule containing all activities of the specified
   personal schedule. In the new personal schedule the execution times of all
   are shifted, so that there is no overlapping with the specified period.
```

The determination of a valid start time is processed as follows. We start with an empty personal schedule. The activities are inserted using earliest-deadline-first, which is in our example activity A with a safe-interval deadline of 14. When inserting the first activity there is obviously no overlap with other activity-executions possible. Therefore the execution of A can be planned at 4, the beginning of A's safe interval, and since its duration is 6 the execution interval for A is scheduled in [4..10].

If there are any overlaps with other execution intervals in the personal schedule the earliest start time in the safe interval without overlaps must be determined. When inserting activity B, the start time of its safe-interval 8 collides with the execution interval [4..10] of A. So it has to be shifted to the end of A's execution interval: 10. Execution of B with duration 10 and starting with 10 is still in the safe-interval [8..20] of B, thus execution of B can be scheduled in the interval [10..20].

No we try to insert activity C after the execution-interval of B, but no non-overlapping execution period can be found in the safe interval and shifting is not possible. So we try to determine a better schedule by shifting the conflicting activity to an earlier start-time. This is demonstrated in our example where the start time of activity C cannot be delayed to resolve the conflict and so the other activities must be moved to earlier start-times, even if that means, that they move out of their safe interval. This is accomplished recursively, until an execution plan without overlaps has been found, which finally results in a personal schedule $PS = \{(A, 0.9, 2, 8), (B, 1.0, 8, 18), (C, 1.0, 18, 30)\}$.

If even recursive shifting of the activities does not result in a non-overlapping execution plan, then the entire personal schedule computation will be aborted. In this case a successful treatment of participants activities is not possible and an escalation decisions must be made quickly, which could be abortion of the activity and its workflow instance, rescheduling of the workflow-instance or other actions as described in [8,9].

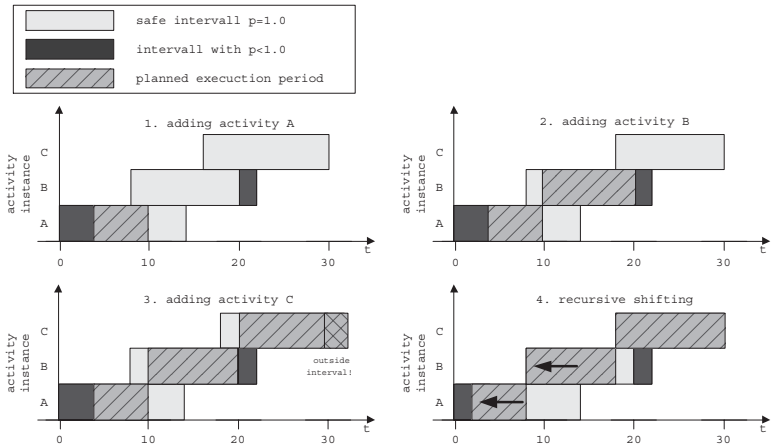

Fig. 6. Example: calculation of a personal schedule

5.1 Admissibility of a Personal Schedule

In our example we could solve problems with overlapping activities by shifting A and B to earlier start-times, but now another problem arises: The resulting personal schedule can no longer be guaranteed, because A is now outside its safe interval, which means that violations of time restrictions may occur. In that case we have to make some statements about how "safe" or "unsafe" a personal schedule is.

A personal schedule is safe, if all its activities can be executed in their planned interval without restrictions. Restrictions arise if an activity is not ready at the planned start time and can not be assigned to the participant. Another reason could be that the scheduled end of the activity leads to deadline violations in the associated workflow instance (the personal schedule "collides" with the time plan). We call this characteristic the *admissibility* of a personal schedule.

Definition 10 (Admissibility of a Personal Schedule) *Let PS be a personal schedule and let p_i be the probabilities of the activities v_i of PS. The admissibility of PS is defined as $\zeta = \prod p_i | (v_i, p_i, e_i, s_i) \in PS$. A personal schedule is safe, iff $\zeta = 1.0$.*

For our example the admissibility can be calculated as: $\zeta = 0.9 * 1.0 * 1.0 = 0.9$, assuming that the execution of every instance activity is an independent event. That means that there is 90% chance that this personal schedule can be met and no time constraint violation, that makes escalation decisions necessary, will occur.

5.2 Workload of a Personal Schedule

Another statement which can be made from the computed personal schedules refers to the workload of participants in certain time intervals. Using a personal schedule it is possible to determine the expected workload for a given period, *i.e.,* for one day or for one week. The workload of a participant within any period can be determined from the overlap of the desired period with all activity entries of the personal schedule. Formally the workload of a personal schedule is defined as:

Definition 11 (Workload of a Personal Schedule) *Let PS_ρ be the personal schedule of participant ρ. The workload of the personal schedule within a period $\gamma = (s, e)$ is defined as:*

$$\eta = \sum overlap(\gamma, (s_i, e_i))|(v_i, p_i, s_i, e_i) \in PS_\rho$$

$overlap((s_1, e_1), (s_2, e_2)) = max(min(e_1, e_2) - max(s_1, s_2), 0)$ *returns the overlap between two time intervals.*

E.g. assuming that the time-units in our example are days, the workload for the first week $(0, 7)$ may be determined as:

$$\eta_{(0,7)} = overlap((0, 7), (2, 8)) + overlap((0, 7), (8, 18)) + overlap((0, 7), (18, 30)) = 5 + 0 + 0 = 5.$$

That means that the participant, for whom our example personal schedule was provided, is busy on 5 out of 7 days in the coming week.

5.3 Minimized Personal Schedule

A minimized personal schedule represents a variant of the original personal schedule, in which we try to shift as much activities as possible out of a certain period without violating time restrictions. This computation makes only sense on on safe personal schedules, since otherwise the minimum workload would be larger than the available time.

By shifting activities out of a certain period, one receives the minimum load for a participant from the workflow in this period and thus a good basis for various planning decisions (*i.e.,* accept additional orders, vacation planning, ...). The computation of a minimized personal schedule is made similar to the original personal schedule calculation. An additional constraint is added: Try to move all activities, whose planned execution time overlaps with the given minimized period, out of this period. Shifting the activities may only take place within their safe interval.

5.4 Maximized Personal Schedule

A maximized personal schedule represents another variant of the original personal schedule, in which we try to shift as much activities as possible in a certain

period without violating time restrictions. The determination of maximized personal schedules makes sense in order to avoid unbalanced workloads. It has to be considered that by shifting the activities their execution can be unnecessarily delayed. Buffer time is lost and cannot be used by following activities if time exceeding occurs.

6 Applications and Conclusions

In the previous sections we introduced probabilistic timed graphs, time plans and personal schedules. Finally we want to describe some applications to demonstrate how this information can be used in workflow systems.

- Provide early information about future activities for participants: At the start of a workflow instance participants can already be informed about their future tasks since this information is contained in the computed time plans.
- Recognize delays because of overload: The admissibility of a personal schedule is limited due to the overloading of the participants. If this admissibility of the personal schedule is monitored, then delays in the instances can be recognized and by observing them over a certain time period bottlenecks can be identified.
- The admissibility of a personal schedule gives information about the probability with which a successful execution of the instances, the participant is involved in, is possible. This value can be linked with threshold values, in order to be able to accomplish automatic control of the current instances. It could be specified that if the admissibility of a workflow instance falls below 95%, a warning is triggered. For critical workflows the value could also be set conservatively to 100%. Further a second value (*i.e.,* 80%) could be specified causing an error alarm to occur if the admissibility of the personal schedule falls below this threshold. Such a model is called *traffic light model* (see [4]) since different states are assigned to each workflow instance according to its admissibility: green, yellow, red.
- Determine overloaded and idle participants: When computing and evaluating the future workload for certain periods (*i.e.,* next week, next month, ...), then it is easy to recognize whether certain participants will be overloaded or will have idle time left. This way the basis for controlling interferences is given. We have to admit that these mechanisms could also be used to observe the efficiency of employees (which was never our intention).
- Before the start of a new workflow instance a personal schedule can be calculated to check if the new instance will lead to some constraint violations due to capacity bottlenecks. If the personal schedule is safe, then the instance can be started without any problems. Otherwise it should be considered whether measures have to be taken in order to ensure a successful execution of all instances, or if a delay of the new instance is necessary. It may be useful to embed the computation and analysis of personal schedules into a system for scenario planning.

– Early warning systems and scenario planning: Future bottlenecks and exceeding of time limits should be recognized as early as possible. An early warning system for workflow systems can be established with personal schedules. Before starting a new workflow instance personal schedules for all participants are computed. By computing the admissibility, the probability for successful execution can be determined, before wasting any time to the new instance. Now one can decide whether the instance has to be started at all, to delay the execution of the instance or perhaps use additional resources to accelerate some tasks.

The main objective of personal schedules is to make information about future tasks available for workflow participants as early as possible and to provide them a preview of their future working schedule. The integration of personal schedules into personal digital assistants, and work-list managers, as well as feedback from personal schedulers to workflow time managers are subject of ongoing research.

References

1. J. Brucker. Scheduling Algorithms. Springer Verlag, 1998.
2. C. Bussler. Workflow Instance Scheduling with Project Management Tools. In *9th Workshop DEXA'98*, 1998. IEEE Computer Society Press.
3. J. Eder and H. Pichler Duration Histograms for Workflow Systems In Proceedings of the Working Conference on Engineering Information Systems in the Internet Context, 2002, Kanazawa, Japan, Kluwer Academic Publishers, page 239-253.
4. J. Eder and E. Panagos. Managing Time in Workflow Systems. In *Workflow Handbook 2001*. Future Strategies INC. in association with Workflow Management Coalition, 2000.
5. J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *Proc. International Conference CAiSE'99*. Springer Verlag, 1999.
6. O. Marjanovic, M. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Syst.*, 1(2), 1999.
7. M. Ninaus Auslastungsberechnungen in probabilistischen Workflow Systemen *Masterthesis*. ISYS Department, University of Klagenfurt, Austria, 2002.
8. E. Panagos and M. Rabinovich. Predictive workflow management. In *Proceedings of the 3rd International Workshop on Next Generation Information Technologies and Systems*, Neve Ilan, ISRAEL, June 1997.
9. E. Panagos and M. Rabinovich. Reducing escalation-related costs in WFMSs. In *NATO Advanced Study Institue on Workflow Management Systems and Interoperability*, Istanbul, Turkey, August 1997.
10. H. Pozewaunig, J. Eder, and W. Liebhart. ePERT: Extending PERT for workflow management systems. In *First European Symposium in Advances in Databases and Information Systems (ADBIS)*, St. Petersburg, Russis, 1997.

A Process-Oriented Model for Authentication on the Basis of a Coloured Petri Net

Peter Lory

Institut für Wirtschaftsinformatik, Universität Regensburg,
D-93040 Regensburg, Germany
Peter.Lory@wiwi.uni-regensburg.de,
http://www.uni-regensburg.de/Fakultaeten/WiWi/lory/

Abstract. Public-key cryptography is a prerequisite for security in distributed systems and for reliable electronic commerce. The protection of public keys against attacks is the Achilles' heel of public-key cryptography. It is the goal of public-key infrastructures to provide the authenticity of the public keys for its participants. Formal models (called trust models) contribute decisively to a deeper understanding of the desirable design principles for these infrastructures. The present paper gives a trust model on the basis of a coloured Petri net. The graphic representation of nets of this type makes them easily understandable even for unexperienced users. In an application in electronic commerce the process formalized by this Petri net will be embedded in a cryptographic protocol which again will be an important part of a larger business process. So, the model of this paper is a useful module in business processes that are common in electronic commerce.

1 Introduction

Public-key cryptography is a prereqisite for electronic commerce and electronic government. Its basic ideas have been introduced by Diffie and Hellman [3] in 1976. A few years later, Rivest, Shamir and Adleman [18] have made an important contribution with their technique that is now well known as RSA-method. For a survey in the basic concepts and methods of public-key cryptography the interested reader is referred to standard references such as [14] or [20].

The two most important applications of public-key cryptography are key management and the generation and verification of digital signatures. The former generates a secret key shared by two entities that have not shared a secret key initially. Once such a key is shared by two entities, it allows the application of the efficient techniques of symmetric cryptography providing confidentiality. The purpose of digital signatures is the provision of authentification, data integrity and non-repudiation.

Public-key cryptography is characterized by its asymmetry. Every entity has a pair of keys: a public key p and its corresponding secret key s. Let p_B and

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 232–245, 2003.

s_B denote Bob's [1] public and secret key, respectively. If Alice wants to send Bob a confidental message, she encrypts the plaintext by his public key p_B. Bob decrypts the arriving ciphertext by his secret key s_B. Conversely, if Bob wants to sign a digital message, he applies his secret key s_B to this message. Alice can use Bob's public key p_B to verify this signature. It is obvious, that Bob must protect his secret key in a personal security environment (e. g. a personal chipcard). Although public keys do not need to be kept secret, and in fact wide knowledge of an entity's public key is desirable, the security problem is that Alice must know for certain that a particular public key really does belong to Bob. If Alice can be tricked into thinking that Mallory's public key is Bob's, Mallory can impersonate Bob to Alice. The protection of public keys against attacks is the vulnerable spot of public-key cryptography. Indeed, citing [23]: *This whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. It is the Achilles' heel of public key cryptography, and a lot of complexity is tied up in solving this one problem.*

It is the goal of a *public-key infrastructure* (PKI) to solve the above mentioned problem. For surveys in this area see [15] and [19]. Public-key infrastructures rest on the concept of a *public-key certificate*. This is a data structure consisting of a data part and a signature part. The data part contains cleartext data including, as a minimum, a public key, a string identifying the entity to be associated therewith and a string identifying the issuer of the certificate. The signature part consists of the digital signature of the issuer over the data part, thereby binding the entity's identity to the specified public key. In practice, certificates obey standards such as X.509 and include additional data fields, for instance the expiration date. If Alice has several certificates, she can build a *chain of certificates* where each public key is certified by the previous entity in the chain, and where she has specified the first public key as authentic and all intermediate entities as trustworthy. Section 2 shows how trust can be propagated in a similar way by the concept of a *recommendation*. Usually, these pieces of information are stored at different places. So, a public key infrastructure can be seen as a distributed database of public-key certificates, recommendations and further information. Thus, it forms a web of certificates and recommendations. Trust plays a prominent role in this web. Consequently, models for public-key infrastructures are often called *trust models*. In contrast to other models for public-key infrastructures (e.g. [4]) the present paper models trust explicitly.

Usually, a user of a public-key infrastructure has a set of statements about the authenticity of certain bindings between public keys and entities and on the trustworthiness of certain entities. Together with the available collection of certificates and recommendations these statements make up the user's (Alice's) view to the public-key infrastructure. It is not the aim of this paper to discuss

[1] Please note, that here and in the following entities are often called Alice and Bob (following tradition in cryptography). However, the reader should keep in mind that they could be a human, a server, a client machine or a personal token like a chipcard or something else.

the problem, how Alice can find the necessary set of information that enables her to prove the authenticity of a certain public key. However, it is obvious that a hierarchical structure or a web of hierarchies can facilitate this procedure.

The present paper focuses on another process: Let the set of statements (Alice's view) be given. The user (Alice) has the aim to derive further statements from this set. For example, let Alice have a certificate that is issued by entity Z and signed by its public key p_Z. Additionally, let Alice have statements on the trustworthiness of Z and on the authenticity of the binding between p_Z and Z. Let the certificate say that p_X is a public key of entity X. Then, these pieces of information enable Alice to derive the authenticity of the binding between p_X and X. If Alice wants to prove the authenticity of a certain binding, then in general she has to perform several steps of this type. Hence, she must carry out a process. This process can be modelled concisely by a coloured Petri net (for details see Section 4). In an application in electronic commerce this process will be embedded in a cryptographic protocol which again will be a part of a larger business process. So, the model of this paper is a useful module in business processes that are common in electronic commerce.

The occurrence graph is an extremely helpful tool in the analysis of the dynamic properties of a given coloured Petri net. Indeed, in the case of the present Petri net it provides all the statements about the authenticity of public keys that can be derived from Alice's view. The model in [13] achieves the same goal by using a logical calculus, from which the present model is derived to a high extent. However, the modelling technique of coloured Petri nets is more easily accessible for users in a business process environment.

The paper is organised as follows: Section 2 gives a precise definition of Alice's view and gives a few examples. In Section 3 the underlying process is discussed in detail. Section 4 defines the coloured Petri net trust model. It focuses on the main aspects of public-key infrastructures and does not yet include certificate revocation. The occurrence graph (Section 5) is the decisive tool for the determination of all the statements about authenticity that are consistent with Alice's view. Concluding remarks are made in Section 6.

2 Alice's View

Certificates are produced with the intention to propagate authenticity of public keys. However, this goal is achieved only if the user of the certificate trusts its issuer. Since the user cannot know personally all the entities he has to rely on, there is also a need for propagation of trust. This task is done by recommendations. Public-key management models that include recommendations have been proposed in [22], [1], [13], [9]. A recommendation can be considered as a signed statement about the trustworthiness of another entity. Consequently, it is similar to a certificate. Nevertheless, recommendations have an additional feature that makes them more complicated than certificates. There exist several levels of trust. Trust of level 1 says, that the trusted entity is trustworthy to issue certificates. A recommendation of level 1 says that the recommended entity can

be trusted with the same level. Proceeding further in a recursive manner, trust of level i means, that the entity can be trusted to issue recommendations of level $i-1$. A recommendation of level i refers to trust of the same level.

Alice as a user sees the web of certificates and recommendations that build the public-key infrastucture. Let it be her aim to establish the authenticity of another person's, for instance Bob's, public key. For this purpose she builds her initial view, which includes all the certificates and recommendations that can be relevant for authenticating Bob's public key and can be retrieved from the public-key infrastructure. The latter means any method of obtaining certificates or recommendations, for instance by accessing a certificate server. Additionally, Alice's view includes statements as a part of her belief, such as authenticity of certain public keys and trust in certain entities. Formally speaking, Alice's view is a set of statements of the type given in the following definition (cf. [13]).

Definition 1 (Statements and Alice's view). Alice's view is a set of statements of the following type:

- *Aut(X,P)* says that Alice is convinced that the public key P belongs to entity X (authenticity).
- *Cert(X,P,Y,Q)* says that Alice holds a certificate, which asserts that Q is a public key for entity Y. This certificate is allegedly issued and signed by entity X. The signature passes verification by the public key P.
- *Trust(X,1)* says that Alice is convinced that entity X is trustworthy of level 1, i.e. this entity can be trusted for issuing certificates.
- *Rec(X,P,Y,i)* says that Alice holds a recommendation of level i for entity Y, i.e. it asserts that entity Y is trustworthy of level i. This recommendation is allegedly issued and signed by entity X. The signature passes verification by the public key P.
- *Trust(X,i)* with $i > 1$ says that Alice is convinced that entity X is trustworthy of level i, i.e this entity can be trusted for issuing recommendations of level $i-1$.

A remark about the word "alleged" in the definitions for certificates and recommendations seems in place: Without verification, it is not clear that entity X has issued the certificate or the recommendation, respectively. However, if Alice can gain evidence that the public key P belongs to entity X, she can verify that entity X is indeed the issuer.

Alice's view allows a graphic representation. Figure 1 gives the graphic elements for the statements of Definition 1. The elements for the statements $Cert(X,P,Y,Q)$ and $Rec(X,P,Y,i)$ nicely illustrate the propagation properties of certificates and recommendations, respectively.

The model of the present paper imposes no restrictions on the underlying architecture. Recommendations and certificates can be issued independently. Indeed, in the following example (from [13]) the chain of certificates and the chain of recommendations follow different paths.

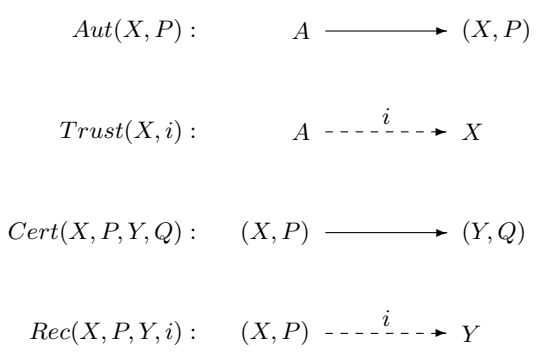

Fig. 1. Graphic elements illustrating Alice's view ("A" refers to Alice)

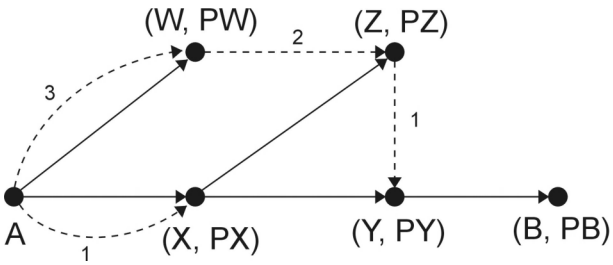

Fig. 2. Alice's view (Example 1)

Example 1. Let Alice's view consist of

$$Aut(X, PX),\ Aut(W, PW),\ Trust(X, 1),\ Trust(W, 3),$$
$$Cert(X, PX, Y, PY),\ Cert(X, PX, Z, PZ),\ Cert(Y, PY, B, PB),$$
$$Rec(W, PW, Z, 2),\ Rec(Z, PZ, Y, 1)\,.$$

Figure 2 illustrates the graph for this view using the the graphic elements given in Figure 1. This example will be continued in Example 3 (see Section 5).

With the exception of node "A" (for Alice) the nodes in the graph of Figure 2 are pairs of entities and public keys. Each pair represents a binding between an entity and its alleged public key. Whether this binding is authentic under Alice's view or not can be decided by applying formal methods (see Sections 3 and 4). The nodes in the corresponding graphs in [13] are entities alone. This reflects the implicit assumption in [13] that every entity controls only one key (in contrast to the present paper); therefore it is not explicitly mentioned which key is concerned. This fact has raised confusion in papers like [16], [17] and has been clarified in [10]. The present paper follows [13], [10] and [11] in modelling trust in persons and not in keys.

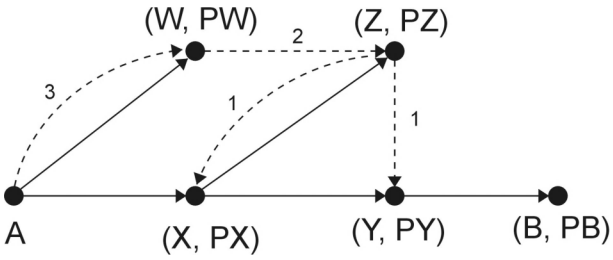

Fig. 3. Alice's view (Example 2)

Example 2. In another example Alice's view consists of

$$Aut(X, PX),\ Aut(W, PW),\ Trust(W, 3),$$
$$Cert(X, PX, Y, PY),\ Cert(X, PX, Z, PZ),\ Cert(Y, PY, B, PB),$$
$$Rec(W, PW, Z, 2),\ Rec(Z, PZ, Y, 1),\ Rec(Z, PZ, X, 1)\,.$$

Figure 3 illustrates the graph for this view. At a first glance this example looks similar to Example 1. Nevertheless, its behaviour is very different (see Example 4 in Section 5).

3 The Underlying Process

As a participant in a public-key infrastructure Alice has a set of statements collected in her view. It is her aim to derive from this set further *Aut-* and *Trust*-statements that are consistent with her original view. The article [13] treats Alice's view as a set of axioms and (similarly to a calculus in propositional logic) gives inference rules for deriving new statements about the authenticity of public keys and the trustworthiness of entities. This calculus allows a straightforward extension to the case where more than one public key may belong to one entity (as is assumed in the present paper). Using this calculus in a top-down fashion the set of derivable statements can be produced. This set forms the closure $\overline{View}$. It contains exactly those statements that are consistent with Alice's (original) view. These statements are called *valid*.

In [21] a LISP-code based on this calculus is given that allows to decide whether a given binding between an entity and a public key is authentic under Alice's view or not. The algorithm starts at this binding and searches in a bottom-up manner a path to the elements of Alice's view through the tree of possible derivations.

The present paper uses the above mentioned top-down approach. However, it does not employ a calculus but formalizes the derivation process by a coloured Petri net (for details see Section 4). Petri nets are ideal tools to model processes. These nets have a graphic representation which makes them easily accessible even for non-experts. Coloured Petri nets have a well-defined semantics which unambiguously defines the behaviour of the net.

The process Alice has to deal with is a result of Alice's interest in *Aut*-statements. Each of these statements proves the authenticity of a binding between a public key and an entity. The other statements, *Trust*, *Cert* and *Rec*, are of no direct value for Alice. Their purpose is to support the derivation of new *Aut*-statements. These derivations have to satisfy certain rules. These rules can be formalized and made precise as transitions in the coloured Petri net. The first rule uses (among other information) a certificate and derives a new *Aut*-statement. The purpose of the second rule is the production of new *Trust*-statements. Consequently, this rule employs a recommendation (among other data). Because all occurrence sequences of the Petri net of Section 4 are finite (see Theorem 1), the occurrence graph can be computed completely. It immediately provides the whole set of valid statements about the authenticity of public keys (see Section 5).

4 The Coloured Petri Net Model

Coloured Petri nets are special high-level Petri nets and have been introduced in [5] where they were called HL-nets. A thorough description is given in the books [6], [7], [8]. The coloured Petri-net of Figure 4 models the process of deriving all the *Aut*-statements that are consistent with Alice's view. It will be explained in detail below. Figure 4 has been drawn by the Design/CPN-software. This is a graphic computer tool which supports the practical use of coloured Petri nets. The paper [12] provides a comprehensive road map to its practical use and gives further references. Resources and technical support on Design/CPN are available via the web site [2]. All simulations in the present paper including the occurrence graph analysis in Section 5 have been performed with this software.

The coloured Petri net of Figure 4 uses seven colour sets (types). They are defined in the global declaration node. The net has four places: `Alice`, `Certificates`, `TrustPool` and `Recommendations`. The place `Alice` acts as a pool of *Aut*-statements. Consequently, its colour set is `Aut`, which is defined as the cartesian product of the colour sets `Entity` and `Keystr`. Thus, a token of this type is a pair of strings. The first string identifies an entity; the second string represents a key (cf. Definition 1). The place `Certificates` collects the *Cert*-statements. These statements can be modelled as quadruples of strings with identifiers for entities in the first and third components and keystrings in the second and fourth components. The colour set (`Cert`) of this place is defined accordingly in the global declaration node. The places `TrustPool` and `Recommendations` act as a pools of *Trust*-statements and *Rec*-statements, respectively. The corresponding colour sets, `Trust` and `Rec`, are again defined in the global declaration node according to Definition 1. The initial marking in Figure 4 corresponds to Alice's view of the Example 1. Please note, that at each place the marking is a multi-set (see [6]) over the colour set attached to the place. For example, the notation `1‘("X","PX") ++ 1‘("W","PW")` means that this multi-set contains one appearance of the token `("X","PX")` and one appear-

```
color Entity = string;
color Keystr = string;
color I = int;
color Aut = product Entity * Keystr;
color Cert = product Entity * Keystr * Entity * Keystr;
color Trust = product Entity * I;
color Rec = product Entity * Keystr * Entity * I;
var ent1, ent2, ent3, ent4 : Entity;
var key1, key2, key3 : Keystr;
var i, j, k : I;
fun min(n:I,m:I) = if n>m then m else n;
```

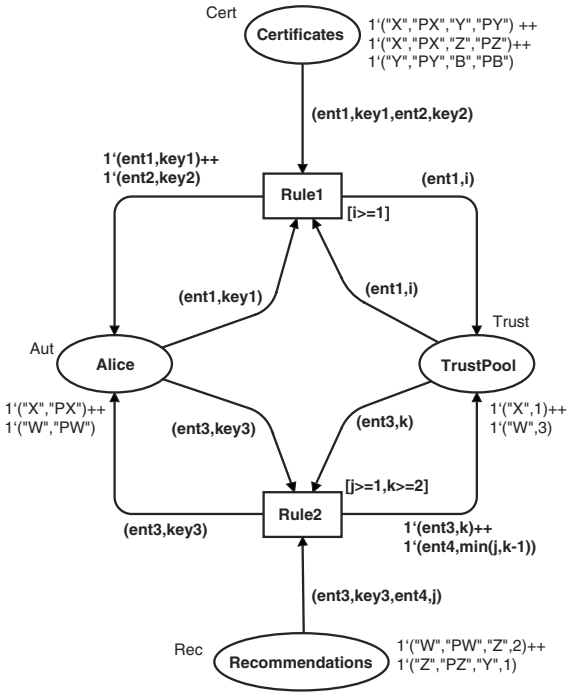

Fig. 4. The coloured Petri net trust model

ance of the token `("W","PW")`. The transitions `Rule1` and `Rule2` are the core of the model. They will now be described in detail.

Rule1: This transition has three incoming and two outgoing arcs and a guard. The variables of this transition are: `ent1` and `ent2` of colour `Entity`, `key1` and `key2` of colour `Keystr`, `i` of colour `I`. Let now data values be assigned to these variables according to Table 1. This creates a *binding* (which should not be confused with the concept of a binding between an entity and a public key).

The pair consisting of a transition and a binding of its variables forms a *binding element*. In order for a binding element to be *enabled* in a certain marking of the places, it must be possible to bind data values to the variables appearing on the surrounding arc expressions and in the guard of this transition such that

Table 1. A possible binding

`ent1` ← `"X"`
`key1` ← `"PX"`
`ent2` ← `"Y"`
`key2` ← `"PY"`
`i` ← 1

1) each of the arc expressions evaluate to tokens which are present in the corresponding input place,
2) the guard is satisfied.

In the case of the binding element

(`Rule1`, binding of Table 1) (1)

these requirements are fulfilled (in the case of the initial marking of Figure 4). If a binding element is enabled, it is ready to *occur*. An occurrence of the binding element (1) removes a token with the values `("X","PX","Y","PY")` from the place `Certificates`, it removes a token with the values `("X","PX")` from the place `Alice` and it removes a token with the values `("X",1)` from the place `TrustPool`. Further, it adds the tokens `("X","PX")` and `("Y","PY")` to the place `Alice` and the token `("X",1)` to the place `TrustPool`. Hence, the occurrence of the binding element (1) has the effect, that the token `("X","PX","Y","PY")` (representing the corresponding *Cert*-statement) is removed from the place `Certificates` (the pool of certificates) and the token `("Y","PY")` (a new *Aut*-statement) is added to the place `Alice`, which acts as the pool of *Aut*-statements. The tokens `("X","PX")` and `("X",1)` return to their places. This is essential, because they may be needed in further steps. It is not necessary to return the token `("X","PX","Y","PY")`. This token acts as a certificate with the only purpose to establish the statement *Aut(Y,PY)* (represented by the token `("Y","PY")`. Once this is done successfully, the certificate cannot be of any further value.

The generalization of this example is straightforward and shows that the transition `Rule1` acts as a producer of statements about the authenticity of public keys. It states that Alice can derive the authenticity of the binding between the entity Y and the public key Q (denoted by *Aut(Y,Q)* and represented in the model by the token `("Y","Q")`), if the following three conditions are satisfied:

1. Alice holds a certificate, which says that Q is a public key for entity Y. The alleged issuer and signer of this certificate is entity X and the signature passes verification by the public key P. This is denoted by *Cert(X,P,Y,Q)* and represented in the model by the token `("X","P","Y","Q")`.
2. Alice has or can derive the authenticity of the binding between entity X and public key P. This is denoted by *Aut(X,P)* and represented in the model by the token `("X","P")`.

3. Alice has or can derive trust of level i with $i \geq 1$ for entity X. This is denoted by *Trust(X,i)* and represented in the model by the token `("X",i)`. It is tacitly assumed that trust of level i implies trust of lower levels.

Rule2: This transition has also three incoming and two outgoing arcs and a guard. The variables of this transition are: `ent3` and `ent4` of colour `Entity`, `key3` and `key4` of colour `Keystr`, `i` and `k` of colour `I`. The reasoning for this transition is similar to that for transition `Rule1`. So, an example is omitted here.

The transition `Rule2` acts as a producer of statements about the trustworthiness of entities. It states that Alice can derive trust in entity Y of level m (denoted by *Trust(Y,m)* and represented in the model by the token `("Y",m)`), if the following three conditions are satisfied:

1. Alice holds a recommendation, which says that Y is trustworthy of level j with $j \geq m$. The alleged issuer and signer of this recommendation is entity X and the signature passes verification by the public key P. This is denoted by *Rec(X,P,Y,j)* and represented in the model by the token `("X","P","Y",j)`. It is tacitly assumed that a recommendation of level j implies recommendations of lower levels.
2. Alice has or can derive the authenticity of the binding between entity X and public key P. This is denoted by *Aut(X,P)* and represented in the model by the token `("X","P")`.
3. Alice has or can derive trust of level k with $k - 1 \geq m$ for entity X. This is denoted by *Trust(X,k)* and represented in the model by the token `("X",k)`. It is again tacitly assumed that trust of level k implies trust of lower levels.

Please note, that the tokens that enter the transition `Rule2` from the places `Alice` and `TrustPool` return to their place. This is essential, because they may be needed in further steps.

The fact that it is not necessary to return the token that enters the transition from the place `Recommendations` requires some consideration: Please note, that the purpose of the transition `Rule2` is the production of tokens of the colour `Trust` on the place `TrustPool`. These tokens correspond directly to the *Trust*-statements that are present in or can be derived from Alice's view. Let for a moment an arc from the transition `Rule2` to the place `Recommendations` be added to the Petri net and let its arc expression be the same as for the arc in the opposite direction. It is easy to see that the second components of those tokens of the colour `Trust` that can be produced in this modified Petri net cannot surpass the second components of the corresponding tokens that can be delivered by the original Petri net of Figure 4. Indeed, let a token on the place `Recommendations` be denoted by `("X","P","Y",j)`. If this token is used in a binding for transition `Rule2` together with the tokens `("X","P")` (on the place `Alice`) and `("X",k)` (on the place `TrustPool`) and the corresponding binding element is enabled, then the token `("Y",m)` with `m = min(j,k-1)` is added to the place `TrustPool`. Let the token `("X","P","Y",j)` be returned to the place `Recommendations` in the modified net. When this token is used again in one of the following steps, it can contribute to the production of a token `("Y",n)` with

`n>m` only if a token `("X",l)` with `l>n` is on the place `TrustPool`. However, the production of this token cannot depend on the presence of the token `("Y",m)` because every occurrence of transition `Rule2` in an enabled binding drops the second component in the token on place `TrustPool`.

Theorem 1. *All occurrence sequences of the coloured Petri net of Figure 4 are finite.*

Proof: Each occurrence of transition `Rule1` in an enabled binding reduces the number of tokens on place `Certificates` by one. The same is true for transition `Rule2` with respect to the place `Recommendations`. So, finally the tokens on the places `Certificates` and `Recommendations` are exhausted.

5 Role of the Occurrence Graph

The prime interest in the application of the coloured Petri net of Figure 4 is to find all the reachable markings of the place `Alice`, because these markings correspond directly to those *Aut*-statements that can be derived from Alice's view, if the marking corresponding to this view is chosen as initial marking. This is closely related to the concept of the occurrence graph. This graph contains a node for each reachable marking and an arc for each occurring binding element (see [6], [7], [8]). Because of Theorem 1 the occurrence graph for the coloured Petri net of Figure 4 is finite. Consequently, the boundedness properties of the Petri net can be investigated using the occurrence graph.

Particularly useful is the *best upper multi-set bound*, which is delivered in the *standard report* of the Design/CPN occurrence graph tool. A multi-set m is defined as an upper multi-set bound for the place p iff $M(p) \leq m$ for all markings M that are reachable from the initial marking. Here, $M(p)$ denotes the marking of place p. The best upper multi-set bound is the smallest multi-set among these bounds. This definition is sound because the occurrence graph for the coloured Petri net of Figure 4 is finite. Indeed, the best upper multi-set bound for the place p is given by the multi-set $\max_{M \in V} M(p)$, where V is the set of nodes (reachable markings) of the occurrence graph (see Proposition 1.13 in [7]). Consequently, the best upper multi-set bound for the place `Alice` in the coloured Petri net of Figure 4 contains exactly those tokens that belong to reachable markings of this place.

Example 3. In the case of Example 1 Design/CPN calculates the best upper multi-set bound for the place `Alice` to

```
1‘("B","PB") ++ 1‘("W","PW") ++ 1‘("X","PX") ++ 1‘("Y","PY") ++
1‘("Z","PZ").
```

This notation for a multi-set follows the output of Design/CPN and has been explained in Section 4. Thus, the result proves the statements

Aut(B,PB), Aut(W,PW), Aut(X,PX), Aut(Y,PY), Aut(Z,PZ).

This example illustrates that recommendations and certificates can be issued independently. For example W has issued a recommendation for Z without certifying Z's public key. It is not even necessary that W knows Z's public key.

Example 4. In the case of Example 2 Design/CPN calculates the best upper multi-set bound for the place `Alice` to `1‘("W","PW") ++ 1‘("X","PX")`, thus proving the statements *Aut(W,PW)* and *Aut(X,PX)*. This shows that the bindings between Y and PY, between Z and PZ and between B and PB cannot be proven. Indeed, if X is dishonest (and this is consistent with Alice's view), X can fake the recommendation *Rec(Z,PZ,X,1)* by using a public key PZ which is the public companion to one of X's private keys. So, X can recommend himself/herself. Additionally, he/she can generate fake certificates *Cert(X,PX,Z,PZ)*, *Cert(X,PX,Y,PY)*, *Cert(Y,PY,B,PB)* and also the recommendation *Rec(Z,PZ,Y,1)*. Consequently, it must be impossible to derive the statements *Aut(Z,PZ)*, *Aut(Y,PY)*, and *Aut(B,PB)*.

Example 5. In a more complex case (from [21]) let Alice's view be

$$Aut(X, PX),\ Aut(F, PF),\ Aut(U, PU),\ Aut(L, PL),\ Aut(Y, PY2),$$
$$Trust(Y, 3),\ Trust(U, 6),\ Trust(L, 36)$$
$$Cert(Y, PY, B, PB),\ Cert(X, PX, Z, PZ),\ Cert(Z, PZ, Y, PY),$$
$$Cert(U, PU, G, PG),\ Cert(X, PX, H, PH),\ Cert(G, PG, W, PW),$$
$$Cert(W, PW, B, PB),\ Cert(L, PL, N, PN),\ Cert(N, PN, B, PB),$$
$$Rec(Y, PY, X, 1),\ Rec(Y, PY, Z, 2),\ Rec(F, PF, X, 2),$$
$$Rec(F, PF, Z, 3),\ Rec(G, PG, F, 8),\ Rec(U, PU, G, 13),$$
$$Rec(U, PU, F, 1),\ Rec(X, PX, H, 2),\ Rec(H, PH, G, 2),$$
$$Rec(F, PF, N, 7).$$

Design/CPN determines in this example the upper multi-set bound for the place `Alice` to

```
1‘("B","PB") ++ 1‘("F","PF") ++ 1‘("G","PG") ++ 1‘("H","PH") ++
1‘("L","PL") ++ 1‘("N","PN") ++ 1‘("U","PU") ++ 1‘("W","PW") ++
1‘("X","PX") ++ 1‘("Y","PY") ++ 1‘("Y","PY2") ++ 1‘("Z","PZ"),
```

thus proving the statements

$$Aut(B, PB),\ Aut(F, PF),\ Aut(G, PG),\ Aut(H, PH),$$
$$Aut(L, PL),\ Aut(N, PN),\ Aut(U, PU),\ Aut(W, PW),$$
$$Aut(X, PX),\ Aut(Y, PY),\ Aut(Y, PY2),\ Aut(Z, PZ)\,.$$

This example clearly demonstrates the usefulness of computer assistance in determining all the derivable *Aut*-statements.

In the examples above, the place `Alice` plays the prominent role, because usually the main interest is in the valid *Aut*-statements. These statements directly correspond to those public keys that are authentic under Alice's view. If the interest is in the *Trust*-statements, the best upper multi-set bound for the place `TrustPool` has to be used.

6 Conclusions

A coloured Petri net has been presented that models a public-key infrastructure. Standard software tools for coloured Petri nets (such as Design/CPN) are directly applicable to this model to determine exactly those bindings between entities and public keys that can be derived from the initially available informations on authenticity of public keys, certificates, recommendations and trust in entities. The use of Petri nets is attractive, because these nets can be understood relatively easily even by unexperienced users. The present Petri net can be integrated into a more complex Petri net modelling a cryptographic protocol.

Trust models can be divided into two classes: deterministic models and probabilistic models. In the former class the statements can be only *valid* or *not valid*, whereas models of the latter class assign confidence values (for instance between 0 and 1) to statements about authenticity and trust (see [1], [13], [9], [10]). In this sense the present model is a deterministic model. Its transformation to a probabilistic model and the incorporation of certificate revocation are topics of further research.

References

1. Beth, T., Borcherding, M., Klein, B.: Valuation of trust in open systems. In: D. Gollmann (ed.): Proceedings 1994 Symposium on Research in Computer Security (ESORICS'94), Lecture Notes in Computer Science, Vol. 875. Springer, Berlin (1994) 3–18
2. Design/CPN online. `http://www.daimi.au.dk/designCPN/`
3. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22** (1976) 644–654
4. Henderson, M., Coulter, R., Dawson, E., Okamoto, E.: Modelling trust structures for public key infrastructures. In: L. Batten and J. Seberry (eds.): Proceedings of the 7th Australian Conference on Information Security Security and Privacy 2002 (ACISP'2002), Lecture Notes in Computer Science, Vol. 2384. Springer, Berlin (2002) 56–70
5. Jensen, K.: High-level Petri nets. In: A. Pagnoni, G. Rozenberg (eds.): Applications and Theory of Petri Nets, Informatik-Fachberichte, Berlin, Vol. 66. Springer, Berlin (1983) 166–180
6. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume I, Basic Concepts. Springer, Berlin (1997)
7. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume II, Analysis Methods. Springer, Berlin (1997)
8. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume III, Practical Use. Springer, Berlin (1997)
9. Jøsang, A.: An algebra for assessing trust in certification chains. In: J. Kochmar (ed.): Proceedings of the Network and Distributed Systems Security Symposium (NDSS'99), Internet Society (1999)
10. Kohlas, R., Maurer, U.: Confidence valuation in a public-key infrastructure based on uncertain evidence. In: Proceedings of the International Workshop on Practice and Theory in Public-Key Cryptography 2000 (PKC'2000), Lecture Notes in Computer Science, Vol. 1751. Springer, Berlin (2000) 93–112

11. Kohlas, R., Maurer, U.: Reasoning about public-key certification: On bindings between entities and public keys. IEEE Journal on Selected Areas in Communication **18** (2000) 591–600
12. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured Petri nets. International Journal on Software Tools for Technology Transfer **2** (1998) 98–132
13. Maurer, U.: Modelling a public-key infrastructure. In: E. Bertino, H. Kurth, G. Martella, and E. Montolivo (eds.): Proceedings 1996 European Symposium on Research in Computer Security (ESORICS'96), Lecture Notes in Computer Science, Vol. 1146. Springer, Berlin (1996) 325–350
14. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton, Florida (1997)
15. R. Perlman: An overview of PKI trust models. *IEEE Network* **13** (1999) 38–43
16. M.K. Reiter and S.G. Stubblebine: Toward acceptable metrics of authentication. In: Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy (1997) 10–20
17. M.K. Reiter and S.G. Stubblebine: Authentication metric analysis and design. ACM Trans. Information and Systems Security **2** (1999) 138–158
18. Rivest, R.L., Shamir, A., Adleman, L: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21** (1978) 120–126
19. Stallings, W.: Network Security Essentials - Applications and Standards. Prentice Hall, Upper Saddle River, New Jersey (2000)
20. Stinson, D.R.: Cryptography: Theory and Practice. CRC Press, Boca Raton, Florida (1995)
21. Wölfl, T.: Automatische Schlüssel-Authentifizierung in einem formalen PKI-Modell. Work done in a student's project, Institut für Wirtschaftsinformatik, Universität Regensburg, 2002
22. Yahalom, R., Klein, B., Beth, T.: Trust relationships in secure systems - a distributed authentication perspective. In: Proceedings of the 1993 IEEE Conference on Research in Security and Privacy (1993) 150–164.
23. Zimmermann, P.: PGP User's Guide, Vol. I: Essential Topics. 1994

Pattern Based Workflow Design Using Reference Nets

Daniel Moldt and Heiko Rölke

University of Hamburg, Computer Science Department
Vogt-Kölln-Str. 30, D-22527 Hamburg
{moldt,roelke}@informatik.uni-hamburg.de

Abstract. The development of workflow applications requires satisfactory concepts and tools. Workflow patterns cover the conceptual part. To base the patterns on high-level Petri nets allows for the tight integration of the modelling editor with the actual execution engine. The development process of workflow applications gains from this.
We propose to use Reference nets as the modelling technique, Renew as the basic execution engine, and our workflow modelling tool for the design of workflows. The latter is a plug-in for the Renew editor, which is based on the use of workflow patterns. The development process is based on prototyping.

Keywords: IDE, high-level Petri nets, nets within nets, patterns, Reference nets, Renew, workflow, workflow patterns

1 Introduction

In the workflow area the topic of patterns has gained some attention. At the workflow pattern page (see [2]) relevant features that have to be covered are listed. Different workflow languages to model business processes exist, implementing various parts of the patterns. In [9] van der Aalst and ter Hofstede present YAWL (Yet Another Workflow Language). The main motivation was that they realized that usual coloured Petri nets are not well suited to model *all* of the workflow patterns.

During the last years our group in Hamburg was involved in several projects which tackled the workflow topics from different sides. First contributions were based on coloured Petri nets as defined by Jensen ([13]) and the nets within nets paradigm (see [14]) in [21] and [7].

A workflow engine supporting a concurrent push/pull architecture for arbitrarily distributed Java clients was developed on the basis of the Renew tool [24]. This workflow engine was used in several places and projects, e.g. in [20] and [19]. A substantial progress has been made by incorporating missing features to extend this tool to a full workflow engine (see [11] and [12]).

The problems mentioned by van der Aalst and ter Hofstede (modelling workflow patterns with coloured Petri nets) were not encountered when modelling workflows with our tool set. Therefore, the question had to be investigated, if these problems generally are solvable using Reference nets ([17]) as a modelling

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 246–260, 2003.

language. The triggering event for this investigation was a talk given by Wil van der Aalst at the CPN Workshop in August, 2002, in Aarhus ([8]).

The goal we had was to show that Reference nets are suitable to model the workflow patterns in an elegant and easy way. The primary concepts we are using here are flexible arcs, net instances, and synchronous channels. Some of the interesting patterns will be shown in this paper. To verify the suitability we developed a plug-in for the Renew Petri net editor to support the construction of workflows by simply putting workflow patterns together in a drag-and-drop manner.

Furthermore the integration into the development process is necessary. Therefore, we discuss how to apply the tool set, which can be seen as a prototype for an integrated development environment for workflow based systems.

It is important to notice that this work does not use the possibility of extending the semantics of Reference nets but only uses given modelling opportunities and carries them forward to a new application area. This is a fundamental difference to the proceeding of van der Aalst and ter Hofstede in [9]. They developed the workflow modelling language YAWL with new constructs and semantics based on Petri nets and transition systems to reach the same goal.

Structure of the paper. First, section 2 introduces some special features of Reference nets not present in other Petri net formalisms. In section 3 some of the criticism of [8] are reproduced and discussed. Alternative modelling approaches using the Reference net formalism are given. Just knowing these workflow pattern net models is not sufficient for modelling larger workflows. Additional tool support is needed, which is sketched in section 4. What may be done to further promote Renew as a workflow development and execution environment is discussed in 5.

2 Special Features of Reference Nets

This section shows some features of Reference nets [16,17] that distinguishes them from other variants of high-level Petri nets. The reader is assumed to be familiar with the concepts of Petri nets in general [22] and coloured Petri nets [13].

Reference nets are an implementation of the nets within nets paradigm of Valk [26,25]. The general idea is that tokens of a so called *system net* are again nets, called *object nets*[1]. Reference nets allow tokens in different places to *reference* the same object net – hence the name. Other variants of nets within nets are possible [26,15].

Some (special) concepts of Reference nets have proven to be very useful for the modelling of workflows: net instances together with synchronous channels and flexible arcs. Because these concepts are not known in many other Petri net

[1] The names emphasise the close relationship between nets within nets and concepts from object oriented programming.

formalisms (and thus in the corresponding editors) we will now shortly introduce them together with other differences.

Net Instances. Net instances are similar to objects of an object oriented programming language. They are instantiated copies of a template net like objects are instances of a class. Different instances of the same net can take different states at the same time and are independent from each other in all respects.

Nets as tokens. Reference nets implement the "nets within nets" paradigm of Valk [25]. This paper follows his nomenclature and denominate the surrounding net *system net* and the token net *object net*. Certainly hierarchies of net within net relationships are permitted, so the denominators depend on the beholders viewpoint.

Synchronous channels. A synchronous channel [10] permits a fusion of transitions (two at a time) for the duration of one occurrence. In Reference nets (see [16]) a channel is identified by its name and its arguments. Channels are directed, i.e. exactly one of the two fused transitions indicates the net instance in which the counterpart of the channel is located. The (counterpart) transition is not a priori restricted in such a way and can correspondingly be addressed from any net instance. The flow of information via a synchronous channel can take place bi-directional and is also possible within one net instance. It is possible to synchronise more than two transitions at a time by inscribing one transition with several synchronous channels.

Figure 6 in section 3.3 shows an example of the usage of a synchronous channel: The examination of a witness (object net) is triggered from the witness examination (system) net.

Arc types. In addition to the usual arc types Reference nets offer *reservation arcs* and *test arcs*. Reservation arcs carry an arrow tip at both endings and reserve a token solely for one occurrence of a transition. They are a short hand notation for two opposite arcs with the same inscription connecting a place and a transition. Test arcs (no arrow tips) do not draw-off a token from a place allowing a token to be tested multiple times simultaneously, even by more than one transition (test on existence). Test arcs are sometimes also known as "read arcs".

Flexible Arcs. Flexible input arcs and flexible output arcs were introduced by Reisig in [23]. They allow multiple tokens to be moved by a single arc. Moreover, the token values and even the number of tokens may vary with the binding of the transition's variables. In Renew, these arcs are indicated by attaching two arrowheads instead of one to the end of the arc.

Figure 1 shows a simple example of a net using (both kinds of) flexible arcs. The leftmost net is the initial state. Transition `t1` is activated and will bind a new vector[2] to the variable `v`.

[2] A vector is a Java container data-type, carrying any number of any object inside. The syntax of the example is not correct (*no such constructor*) to make the example more readable.

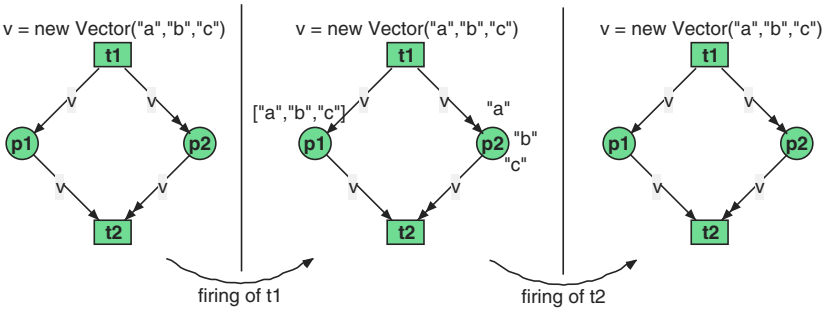

Fig. 1. Flexible arc example net

The centre net shows the state after firing t1: The (normal) arc from t1 to p1 has put the vector v to the place p1 as one object. The flexible arc from t1 to p2 has unpacked the vector and put all contained objects as single tokens to the place – p2 is now marked with three single tokens.

The firing of transition t2 clears both places: t2 is only activated if all objects in the vector v are tokens of place p2. If this matches, all these tokens are removed.

Note that flexible arcs are a part of the concurrent simulator of Renew, i.e. they do not destroy the Reference nets' concurrency semantics. Further information can be found in the original work of Olaf Kummer [17]. Flexible arcs support different container data types: arrays (original implementation), lists, collections (a huge group of Java built-in data types) and some more.

Reference Net Workshop

The Reference Net Workshop (in short: Renew) [18,24] is an integrated development and simulation environment for Reference nets. It fully supports the nets within nets idea: The modelling process creates templates of nets. In the simulation it is possible to generate an arbitrary number of copies of the net templates (each with its own local behaviour), called net instances. This is done in the usual way as tokens are generated in other Petri net formalisms. Renew is written entirely in the programming language Java and uses Java for the inscriptions of the Reference nets. It is therefore possible to integrate any Java object as a token in a Reference net, to call Java object methods from within a net, and to call a net (via a synchronous channel) from within a Java object.

Renew's programming sources are available free of charge (see the web page [24]) and are quite well documented. It is therefore possible to integrate extensions both to the simulation engine (net formalism extensions) as well as to the development environment (new modelling features). A new version of Renew – at the time of writing of this paper not yet released – simplifies the creation of

plug-ins for Renew. This new feature makes it especially easy to add or restrict modelling possibilities to the Renew editor. This allows for the development of editors for special Petri net constructs like workflows (see section 4).

3 Reference Net Workflow Patterns

In this section the arguments of van der Aalst and ter Hofstede of the publication "On the expressive power of Petri net based workflow languages" [8] are repeated and discussed.[3]

Three main problems of Petri net workflow models are identified:

1. No specific support for multiple instances.
2. Or-splits and or-joins are difficult to handle.
3. Non-local effects (e.g. cancellation) are not part of the Petri net formalism.

As we will see throughout this section, all three points of criticism can be solved or at least alleviated by specific constructs of Reference nets. This will be shown on selected examples.

3.1 Workflow Patterns

In the paper mentioned above, van der Aalst and ter Hofstede argue that many workflow patterns can easily – almost trivially – be expressed in Petri nets. Examples for this kind are sequences, and-splits, and and-joins. Other patterns have no intuitive representation when using traditional coloured Petri nets: They require extra net constructs that draw off the attention from the essentials that the pattern should express. The workflow patterns that are dealt with in the paper are:

I. Basic Control Flow Patterns
   1) Sequence
   2) Parallel Split (and)
   3) Synchronisation (parallel, and)
   4) Exclusive Choice (xor)
   5) Simple Merge (exclusive, xor)

II. Advanced Branching and Synchronisation Patterns
   6) *Multi-choice (or)*
   7) *Synchronising Merge (or)*
   8) Multi-merge
   9) *Discriminator*

III. Structural Patterns
   10) Arbitrary Cycles (loop)
   11) *Implicit Termination*

IV. Patterns involving Multiple Instances
   12) Multiple Instances Without Synchronisation (fork)
   13) Multiple Instance With a Priori Design Time Knowledge

[3] The argumentation of [8] is taken up in other papers, e.g. in the YAWL report [9].

14) *Multiple Instance With a Priori Runtime Knowledge*
15) *Multiple Instance Without a Priori Runtime Knowledge*

V. State-based Patterns
16) Deferred Choice (implicit choice)
17) Interleaved Parallel Routing
18) Milestone

VI. Cancellation Patterns
19) *Cancel Activity*
20) *Cancel Case (Instance)*

All patterns in *italic* letters are seen as problematic to model using coloured Petri nets. We will exemplarily show the modelling problems as given by van der Aalst and ter Hofstede and then introduce our solution. As one can see from the enumeration above, the basic control flow patterns pose no problems, so we can start with the advanced control patterns.

3.2 Advanced Branching and Synchronisation

Petri nets do not offer the construction of a "maybe" split or join. It has to be explicitly stated if exactly one out of n possibilities should be selected or synchronised (modelled by a place) or all possibilities should be carried out (in parallel; split and join via transitions). There is no such elementary construct that can decide at runtime which input and output arcs are valid: "maybe there is/should be a token or maybe not". However, in workflows sometimes the construct of Fig. 2 is desirable.

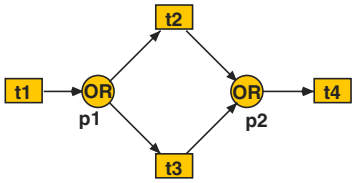

Fig. 2. Or-Split and Join (no Petri net semantics)

Fig. 2 has the following intended meaning: In t1 the decision is made whether t2, t3 or both should be carried out. This decision is made at runtime, i.e. all cases are possible. Because this has no mapping to ordinary Petri nets, the figure cannot express the exact meaning. The next problem is the join after t2 and t3. Petri nets do not allow a simple model of the desired behaviour, which is that following transitions (tasks) should only be activated once, no matter if only one or both of t2 and t3 have fired.

There are some possibilities to model "around" these problems. Two different models are sketched in [8]: passing information from split to join or sending

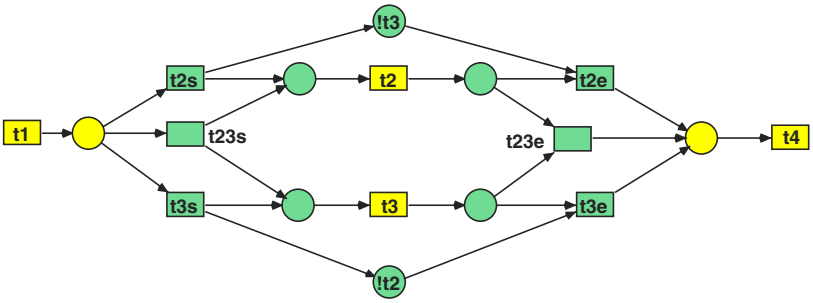

Fig. 3. First proposal: or-split and or-join (v. d. Aalst et al.)

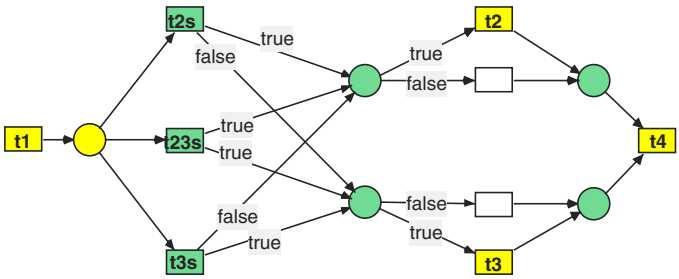

Fig. 4. Second proposal: or-split and or-join (v. d. Aalst et al.)

true/false tokens to each possible selection (t2 and t3 in the example) each task may only be carried out on a "true".

Figure 3 and 4 show models of these two proposals.[4] It is easy to see that the ratio of model elements from the idealised case (Fig. 2) and the additional net elements to model the selection and synchronisation management is quite unsatisfactory.

To our opinion this modelling overhead is the biggest problem with the patterns above.[5]. The modelling overhead has to be spent to enumerate all possible selections. This effort grows exponentially with the number of actions that may be executed – in the examples only two possibilities are given to keep the models simple, but in real-world workflows there is no reason to limit the tasks by this number. (Of course any number of possible actions can be selected by a tree of

[4] From now on, all net figures are valid Reference nets.

[5] Another inconvenience is the information passed from the split construct to the join node. This information passing should be automatically handled by the workflow modelling tool.

several (at least log n) or-splits and -joins with two possibilities each, but such constructs are really confusing and even harder to operate).

This is the situation when the flexible arc comes into play. The flexible arc allows to put an arbitrary number of tokens (even zero, if desired) onto a place where the number of tokens has to be known in advance – but may change from firing to firing at run-time. This is exactly what we need to allow for an elegant model of the or-split and join.

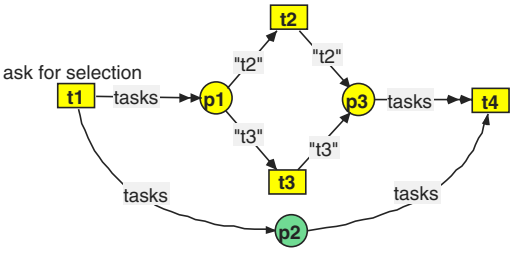

Fig. 5. Or-split and or-join using flexible arcs

In Figure 5 a possible model is shown. The selection transition `t1` prompts the user for information about which tasks should be carried out (this is not shown in the model, it may for example be done using a graphical user interface window) and combines the selection in a data type container (e.g. an array). The selection has to be known at the join node, so it is passed over to place `p2`. The flexible arc from `t1` to `p1` unpacks the array and puts the appropriate number of tokens (zero to two) to the place. The tasks are identified by their names, only those tasks that have been selected may be carried out. After completion of the tasks tokens are put on place `t3`. If the tokens on this place match the content of the container object on place `t2`, transition `t4` is activated and may conclude the split-join construct. Note that no tokens are left on any of the places.

The enlargement of this pattern to more than two alternatives is absolutely straight-forward, only transitions for the alternatives (like `t2` or `t3`) have to be added and the selection process has to be adapted.

3.3 Patterns Involving Multiple Instances

When it comes to multiple instances, Reference nets naturally have a home game. As stated in the introduction, instances of net schemas are one of their basic modelling constructs. It is therefore not surprising that the modeller does not have to deal with hand coded reference counting or other hassles.

Figure 6 shows a possibility to model a simple witness examining example. For each case zero or more witnesses may register. While the case is open, additional witnesses may appear and also have to be examined. The case may only be closed if all registered witnesses have been examined.

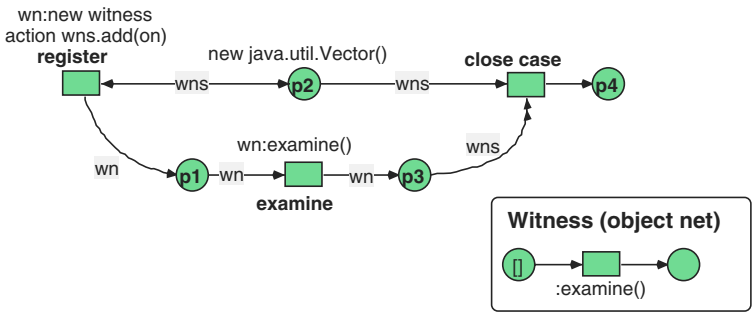

Fig. 6. Example for multiple Instances: witness examination

This is done as follows: For each new case the main net of Fig. 6 is instantiated once. Witnesses are handled as object nets (lower right corner of the figure) of such an object net for a single case – at least a three step hierarchy is built.

Transition register instantiates a new object net of type witness describing the examination of a witness. This object net is bound to variable wn and put on place p1. The reference to the object net is also added to the vector of witnesses on place p2. Transition examine synchronises with the witness object nets while they are examined and moves them to place p3. Only if all witnesses have been examined the case may close (transition close case). This is again ensured by a flexible arc comparing the vector of all witnesses (place p2) with the witnesses already processed (place p3). No new witness can register after this event.

3.4 Cancellation Patterns

Cancellation is a general problem in concurrent systems, not only in the workflow area. Using Reference nets as a modelling language, two approaches are possible: bookkeeping of tokens in extra places (1:1) or use of encapsulated object net tokens with restricted interfaces.

In [8] only bookkeeping is considered. The general statement of the paper is that removing tokens like a vacuum cleaner is too complex due to the exponentially growing number of possible states. However, keeping track of tokens is not that complicated using the flexible arc concept of Reference nets. For each place that has to be considered, an additional place is needed containing the marking information. A cancel activity can now remove the tokens from the original places together with the extra tokens using the flexible arc concept. The effort for doing so is only linear to the number of places.

The cancellation of whole (sub-)tasks is even easier following a strict encapsulation approach: A task is modelled as an object net and put into a special hosting place for such subtasks. The hosting place offers synchronous channels to the task object net enabling it to interact with the outside world. The subtask may not have other communication possibilities than these channels. If the tasks

should be deactivated, it only has to be moved to another place not offering any channels, for cancellation (deletion) the token has to be removed. The object net can possibly carry on firing internally but for any (external) user of the workflow system it disappeared. It is possible to fold all needed communication channels to one general channel.

Figure 7 illustrates a combination of both ideas (bookkeeping and encapsulated net tokens). The net structure is quite similar to the example of the preceding subsection (Fig. 6).

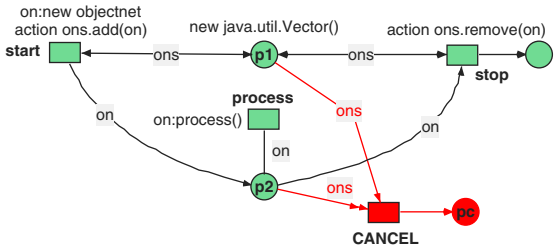

Fig. 7. Cancellation using bookkeeping and instances

New object nets are instantiated by transition start and put on place p2. The vector on place p1 is used for keeping track of the active object nets that may be workflow (sub-)tasks of any type. These object nets can be stopped by transition stop, doing so removes them from the vector. While the object nets are on place p2, internal processing may take place. The communication channels mentioned above are illustrated by the channel on:process at transition process. In the object nets (on) not shown in the figure transitions have to be inscribed with the same channel. The transition CANCEL removes *all* tokens of p1 and p2. No further activity is possible.

4 Renew as a Workflow IDE

Integrated development environments (IDEs) can be characterised as tools that provide a well adjusted set of features to allow to produce code. We propose, on top of the conceptual framework, a tight integration of the build-time and run-time part of workflow systems. The components are tools for modelling and designing workflows and the deployment and enactment tools. In this section we will explain how these are combined and what the related development processes are.

The process design and definition[6] is usually separated from the process instantiation and control. We want to integrate them. In [12] a stable version

[6] In this text we will follow the terminology published in [3] from the Workflow Management Coalition (Wfmc) (see [1]).

of a workflow engine has been introduced for Reference nets. It contains all necessary features for the workflow enactment service. Due to a tight coupling to Java any Java accessible application can be connected.

The proposal here is to extend this environment by the workflow patterns described above and our workflow modelling tool. This allows users to work on a more abstract level. To explain our approach we present a short scenario.

Basic WFMS Technology. Assume e.g. a shop that is based on a workflow system (WFS). Each relevant business event is handled by the WFS. For this system the Workflow Management System (WFMS) presented in [12] is used. Therefore, we have a push/pull architecture. Tasks can be assigned to roles or users. Users can ask for a task which can be selected from a pool of tasks, which again is only visible for authorised persons, supervised by a powerful access control. The whole system can be distributed in that sense, that there is a server which allows for high persistence of the running workflows.[7] Whenever e.g. a power failure occurs the system will restart at the same execution step. Every committed action is made persistent as long as this is supported by the underlying database (see [11]). The smallest recordable action is the firing of single transitions, so fully persistent Petri net workflows are possible. The performance will be reduced dependent on the underlying hardware. No persistence allows for faster execution, however, the recovery performance is bad.[8] Communication is based on encryption as strong as necessary. Therefore, the organisational aspects seem more interesting in this respect. This topic will be picked up again later.

New tool setting. Conventional Workflow tool settings are assuming that there is only a (fixed) run-time version of the WFS. Our new setting provides for the augmentation of new workflows to the system at any time. This is done with the same tools that are used for the enactment. New workflows are modelled using the patterns introduced above. The specific annotations for the application are essential information that have to be added to reach the desired functionality. Such annotations are usually net inscriptions, in our case Java statements. More advanced Java packages can easily be used within Renew. Therefore, any graphi-

[7] According to [3] there can be several servers (=workflow engines) which are connected via a network and can be considered as one logical unit: "Standards to support this transfer of workflow control enable the development of composite workflow applications using several different workflow products operating together as a single logical entity." ([3] p.9) In [7] an enactment concept of inter-organisational workflows based on Reference nets has been introduced.

[8] Our experience so far is that the performance is reduced by a factor of 10-100 depending on the system configuration. Parameters are:

- Communication speed in the distributed system for database, server and clients.
- Visual output on the server for the workflows.
- The speed of the database.
- The overall architecture of the whole distributed workflow system.

cal interface, database, or other functionality that is wrapped by or implemented in Java code may directly be included. This allows for a fast modelling.

Integration of the new workflows is made by the provision of Reference net templates. These can be instantiated at run-time. The WFS only needs to know that there is a new kind of workflow. Workflow names are put in the database and therefore are instantly available if the according files have been deployed in the system.[9]

Besides the deployment and enactment of new workflows often the modification of workflows is necessary. Two cases have to be considered: a) Only new workflows have to be modified. b) Already running workflows have to be modified. The first case is easy: The according Reference net template is modified and added to the database of existing workflows. Any new instance of the net template (=workflow) is then of the new kind.

The second case needs some further discussion: Since the structure of already existing net instances can not be manipulated in Renew a different concept is needed. Especially in distributed workflow systems the workflow definition can be split into several parts: A general frame which contains the specific sub workflows. The smallest units of such sub workflows are the single actions within the system. The proposal is to modify the sub workflows. This can be done if only those sub workflows are instantiated that reflect the actual state of the system. Within the Renew environment this is the natural view.

The last solution also indicates the flexibility that can be reached when designing workflows. A drawback is that at least one additional level of indirection is introduced.

Our integrated development environment (IDE) for workflows allows for a fast prototyping oriented approach. The concept of sub workflows is essential for this. Workflow patterns allow for a structured view. The concept of sub workflows supports abstraction. Since Renew allows arbitrary levels of nesting of Reference nets any finite number of abstraction levels can be reached for the workflows.

The nesting also supports the separation of the workflows into separated parts. An extreme version would be to consider each atomic action as a workflow. The other end would be to consider each workflow as one part. However, this would lead to the above mentioned problem to modify the workflow at run-time. Therefore, the level of abstraction is determined by the need of the modifications of parts of the workflow.

Development Process

When tackling the problem of designing workflows for a business event two approaches are common: bottom-up and top-down. Assume that several workflows already exist. These workflows consist of a certain number of sub workflows. Due to the splitting of workflows some sub workflows may have been already used in several workflows. Starting from these already existing workflows new workflows can be composed (bottom-up). Following the top-down approach new workflows

[9] This again shows the necessary access control within a well designed organisational setting. Otherwise arbitrary tasks can be performed in the system.

can be constructed by refining existing ones and adding new sub workflows accordingly.

Both ways of workflow design can be used as it is appropriate in a given situation. The addition could be done in an arbitrary way, however, we propose to have specific operations which are coupled to the patterns. Analysis of such restricted models is easier than dealing with arbitrary models. The problem of the (Java) inscriptions makes the analysis difficult anyway. However, when just looking at some building blocks which are checked for their correctness and later on analysing only the net structure based on results and tools (see [6], [4] or [5]), this is of real advantage for the designers (and the users).

5 Discussion and Outlook

Our first goal was to show that Reference nets are suitable to model workflow patterns in an elegant and easy way. The patterns not modelled in this paper have also been investigated which is reflected by the fact that they can be found in our workflow modelling plug-in for Renew. Due to the availability of a workflow engine and the tight coupling of that with the modelling tool an integrated development and enactment environment for workflows is easy to implement.

The three main problems mentioned by van der Aalst and ter Hofstede may all be solved with the usual Reference nets as defined in [17]. This was possible due to the concepts of net instances, references to nets, synchronous channels, and flexible arcs. For example we only have a linear complexity when handling the cancellation pattern. It is important to notice that we did not extend the reference formalism for the modelling concepts.

However, we had to put quite some effort (about two person years) into the development of the workflow tool set all together, that is now supplemented with the pattern oriented editor plug-in. The persistence and the workflow engine were the main burden. The development of the pattern modelling plug-in was relatively easy due to the development of another tool which introduced the concept of plug-ins for Renew.

Interesting may be that we can control any kind of Java application.[10]

Only rudimentary work has been done on the connection to analysis tools. Two simple interfaces for Woflan and the Maria tool are part of Renew. However, no deep investigations have been made on the integration into the development process.

For the final practical implementation within the tool set some control is necessary to allow only privileged users to modify or add new (kinds of) workflows. Otherwise there is the risk of misuse when actually deploying an application.

Another interesting feature is the possibility of inter-organisational workflow handling as described in [7]. Everything is prepared, however, no practical experience has been made up to now. Here we expect to face new problems with respect to security, performance, and may be new problems on the conceptual side. Mobile workflows which are an obvious concept in this setting will of course introduce the usual problems of mobile code.

[10] If the Java native interface is implemented it can even be any accessible application.

Last but not least new kinds of workflow patterns will appear in any case, since the development in this sector has not reached its end so far.

References

1. Workflow management coalition homepage. URL: `http://www.wfmc.org/`, 2003.
2. Workflow pattern homepage.
   URL: `http://tmitwww.tm.tue.nl/research/patterns/`, 2003.
3. Workflow reference model.
   URL: `http://www.wfmc.org/standards/docs/tc003v11.pdf`, 2003.
4. Wil M.P. van der Aalst. Verification of workflow nets. Number 1248 of Lecture Notes in Computer Science, pages 407–426, Springer Verlag, Berlin, 1997.
5. Wil M.P. van der Aalst. WOFLAN: A Petri-net-based workflow analyser. In *International Conference on Application and Theory of Petri Nets in Lisbon 1998*, number 1420 in Lecture Notes in Computer Science, Springer Verlag, Berlin, 1998.
6. Wil M.P. van der Aalst, Jörg Desel, and Andreas Oberweis (Eds.). *Business Process Management: Models, Techniques, and Empirical Studies.* Number 1806 in Lecture Notes in Computer Science. Springer Verlag, Berlin, 2000.
7. Wil M.P. van der Aalst, Daniel Moldt, Rüdiger Valk, and Frank Wienberg. Enacting Interorganizational Workflows Using Nets in Nets. In J. Becker, M. zur Mühlen, and M. Rosemann (Eds.), *Proceedings of the 1999 Workflow Management Conference Workflow-based Applications*, Working Paper Series of the Department of Information Systems, pages 117–136, University of Münster, Department of Information Systems, Münster, 1999. Working Paper No. 70.
8. Wil M.P. van der Aalst and Arthur H.M. ter Hofstede. Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages. In Kurt Jensen (Ed.), *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, volume 560 of *DAIMI*, pages 1–20, Aarhus, Denmark, August 2002. University of Aarhus.
9. Wil M.P. van der Aalst and Arthur H.M. ter Hofstede. YAWL: Yet Another Workflow Language. QUT Technical report, FIT-TR-2002-06, Queensland University of Technology, Brisbane, 2002.
10. Søren Christensen and Niels Damgaard Hansen. Coloured Petri nets extended with channels for synchronous communication. In Robert Valette (Ed.), *Application and Theory of Petri Nets 1994, Proc. of 15th Intern. Conf. Zaragoza, Spain, June 1994*, LNCS, pages 159–178, Springer Verlag, Berlin, June 1994.
11. Thomas Jacob, Olaf Kummer, and Daniel Moldt. Persistent Petri Net Execution. *Petri Net Newsletter*, 61:18–26, October 2001.
12. Thomas Jacob, Olaf Kummer, Daniel Moldt, and Ulrich Ultes-Nitsche. Implementation of Workflow Systems using Reference Nets – Security and Operability Aspects. pages 139–154, Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark, 2002. Computer Science Department, University of Aarhus. see also: `http://www.daimi.au.dk/CPnets/workshop02/cpn/papers/`.
13. Kurt Jensen. *Coloured Petri Nets: Volume 1; Basic Concepts, Analysis Methods and Practical Use.* EATCS Monographs on Theoretical Computer Science. Springer Verlag, Berlin, 1992.
14. Eike Jessen and Rüdiger Valk. *Rechensysteme; Grundlagen der Modellbildung.* Springer Verlag, Berlin, 1987.

15. Michael Köhler and Heiko Rölke. Mobile object net systems: Concurrency and mobility. In H.-D. Burkhard, L. Czaja, G. Lindemann, A. Skowron, and P. Starke (Eds.), *Proceedings of the International Workshop on Concurrency, Specification, and Programming (CS&P 2002)*, 2002.
16. Olaf Kummer. Simulating synchronous channels and net instances. In J. Desel, P. Kemper, E. Kindler, and A. Oberweis (Eds.), *Forschungsbericht Nr. 694: 5. Workshop Algorithmen und Werkzeuge für Petrinetze*, pages 73–78. University of Dortmund, Computer Science Department, 1998.
17. Olaf Kummer. *Referenznetze*. Dissertation, University of Hamburg, Computer Science Department, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 2002.
18. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. *Renew - User Guide*. University of Hamburg, Computer Science Department, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 1.6 edition, 2002.
19. Stefan Müller-Wilken. *Mobile Geräte in verteilten Anwendungsumgebungen: Ein Integrationsansatz zwischen Abstraktion und Migration*. Dissertation, University of Hamburg, Computer Science Department, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 2002.
20. Stefan Müller-Wilken, Frank Wienberg, and Wilfried Lamersdorf. On integrating mobile devices into a workflow management scenario. In A. Al-Zobaidie, A. M. Tjoa, and R. R. Wagner (Eds.), *Proc. 11th International Workshop on Database and Expert Systems Applications (DEXA 2000)*, pages 186–192, Hamburg, October 2000. IEEE Computer Society.
21. Daniel Moldt. Using workflows to structure systems based on object-oriented coloured Petri nets. In J. Desel, A. Oberweis, W. Reisig, and G. Rozenberg (Eds.), *Petri Nets and Business Process Management*, number 217 in Dagstuhl Seminar-Report, Saarbrücken, 1998.
22. Wolfgang Reisig. *Petri Nets: An Introduction*. Springer Verlag, Berlin, 1985.
23. Wolfgang Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80(1–2):1–34, 1991.
24. Renew homepage. URL: `http://www.renew.de`, University of Hamburg, Department for Computer Science, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 2003.
25. Rüdiger Valk. Concurrency in communicating object Petri nets. In G. Agha, F. De Cindio, and G. Rozenberg (Eds.), *Concurrent Object-Oriented Programming and Petri Nets*, number 2001 of Lecture Notes in Computer Science, Springer Verlag, Berlin, 2001.
26. Rüdiger Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In Jörg Desel (Ed.), *19th International Conference on Application and Theory of Petri nets*, number 1420 of Lecture Notes in Computer Science, Springer Verlag, Berlin, 1998.

A Model for Process Service Interaction

Karim Baïna[1], Samir Tata[2], and Khalid Benali[1]

[1] LORIA-INRIA-CNRS (UMR 7503)
Campus Scientifique B.P. 239 54506 Vanœuvre-lès-Nancy - FRANCE
{baina,benali}@loria.fr

[2] Institut National des Télécommunications
9, rue Charles Fourier 91011 Evry Cedex - FRANCE
Samir.Tata@int-evry.fr

Abstract. The design and the achievement of any consequent project imply the involvement of several people, teams and even enterprises. These enterprises interact and exchange data, more and more often, through Internet and the "Web". However, exchanging data is not enough for working together, it is also necessary to control and manage these exchanges occurring within business processes. Cooperation between enterprises means interconnecting and coordinating their business processes. If a wide spectrum of tools for work coordination exists, they have been unfortunately developed to only suit the internal needs of enterprises. Thus, existing work coordination systems are not adapted to inter-enterprise cooperation. This paper presents a promising approach for interconnecting processes based on service interaction. Its aim is to formally present a model for enterprise process interconnection and coordination through service interaction based on information sharing between process services, and process services coordination.

1 Introduction

Due to business process automation development, process interconnection and coordination has become an important matter. If a large number of business process management systems exist (e.g. workflows, shared agendas, project managers, to do lists), they have been mainly developed to suit internal needs of enterprises.

In workflow management systems (WFMS), existing interconnection and/or coordination solutions are mostly static and depend on specific business process definition languages and WFMS platforms and with regards to private data exchange formats, etc. To improve generic process interconnection support within existing WFMS, related interconnection models deal with awareness and dataflow formalization between inter-leaving processes (*e.g.* shared dataspace models [17], message passing mechanisms [3], event subscription and notification paradigms [11], remote object invocation [24,16], transfer protocol extension [4]), or with inter-leaving process control (*e.g.* transactional protocols [18,8,2]).

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 261–275, 2003.

Another way to connect and coordinate enterprise processes is based on the web. For instance, several architectures based on electronic service exchange between applications exist (e.g. IONA Orbix E2A [13], Sun ONE [19], Microsoft .NET [15]). However, current specification of Service Oriented Architectures (SOA), based on service definition language (WSDL), service discovery registries [22,12] and service invocation protocol (SOAP)[23], are still in the beginning of their progress. Actually, web services frameworks are not enough mature to support enterprise process interconnection. For instance, SOA lacks a lot of rich mechanisms like service matching, service negotiation and contracting, service sessions, service transactions, service licensing, service security, service communication and coordination, etc.

In our opinion, the more promising and the more generic approach for interconnecting processes should be service oriented. In other words, to develop a model for enterprise process interconnection we can use process service interaction. The service concept has been defined in many research fields: object oriented research [16], process modeling research [10,6], distributed system research, etc. In the field of workflow, a process service may be seen as a software entity which is able to present process particularities and outcomes without totally revealing its structure (i.e. its implantation in a WFMS or project management tool). A process service offers functional abstraction of a process (or a sub-process) supplied by an enterprise or an enterprise accepts to fulfill with a predefined quality of service. A process service specifies the work load that an enterprise accept to fulfill with a predefined quality of service. Process service concept has been studied from several points of view : abstract semantics of process service execution, sub-process service selection, activity coordination of dynamic process service, process service control flow abstraction, process service methods and events wrapping [5], and process service composition [9]. For us, a process service is seen as a pattern supporting cooperation and dynamic interconnection between enterprise processes.

Since we beleve that a service oriented approache is a promising approach and since the SOA approach lak a lot of rich mechanisms, we are developing a general service oriented approach with the following models :

- a negotiation model for computer supported cooperative work we have developed in [14] and we have applied to process services to improve SOA facilities and dynamics [1],
- a model for dynamic selection and interconnection of process service contracts we have developed in [1], and
- a model for process service interaction we present in this paper.

While our previous works dealed with atomic process servicesa [1], this work focuses on composition, cooperation and interaction of process services for enterprises process interconnection. Trying to go further limits of SOA, we present a generic process service interaction model for interconnection of enterprise processes. We will start, in section 2, by defining our process service approach which is at the heart of our process service interaction model. Then we will present the two dimensions on which our process service interaction model is based on :

information sharing between process services in section 3, and process services coordination in section 4. Section 5 will sum up our interaction model. And finally, section 6 will give some concluding remarks.

2 Process Interconnection through Process Service Interaction

Existing process management systems often consider process instance as a black box (process method calls, execution events, and intermediate data are not visible from outside). This approach does not suit process cooperation needs. One of the aim of our process service interaction model is to allow processes to cooperate through partial visibility of their resources (e.g. data sharing, group awareness) by permitting access to a set of their own methods and events [1].

Service exchange is a high level cooperation paradigm. It's based upon existing cooperation paradigms : *production* (dataspace sharing), *communication* (common message format, remote method invocation, notification messages, group awareness), and *coordination* (synchronization, coherence criteria checking). Therefore, service exchange offers a better expressiveness and comprehensiveness for cooperation situations. Our aim is to develop a service structure pattern supporting cross organizational processes cooperation.

A process service can represent either the description of a task that an enterprise may wish to out-source (e.g. because of time lack, cost reducing, or skill needs) or a specific task that an enterprise is known to be skilled to accomplish. This task may be managed by an automatic, a semi-automatic, or a manual process. It may be performable by a unique "provider" (atomic process service), or may need cooperation of several providers (composite process service) (e.g. because it needs several skills not necessary within the scope of a single provider). We distinguish thus between two types of process services : atomic process service and composite process service. The atomic process service represents the interface of a process, which its structure is hidden. The composite process service is constituted by a set of process services which may be themselves atomic or composite. A requester can compose the needed process services by using process services supplied by several providers. To do that, the requester needs a set tools to describe and manage interactions of the different provided process services. The interaction model we propose in this paper takes place in this context. It focus on process service coordination by data sharing and control flow management. We define our process service interaction model as 5 sets :

- a set of access contracts on shared data,
- a set of visible method execution contracts,
- a set of visible event reception contracts,
- a set of coordination contracts based on shared data states,
- a set of coordination contracts based on process service states.

To illustrate our approach, figure 1 shows an example of process service interconnection within an e-learning context. Let *KManager* (an e-learning enterprise)

be a service requester enterprise. Let *WAgency* (a web agency enterprise), *CoolHost* (a site hosting enterprise), and *e-Store* (an e-learning content collection enterprise) be three (among others) service provider enterprises. These enterprises cooperate by providing (and requesting) process services to be achieved within their cooperation. Process services represent either their skills ans requirements within their cooperation partnership. The enterprise *KManager* is the main actor of this cooperation and its aim is to produce e-learning sessions. To do so, it uses process services provided by its partners. Figure 1 shows how our model can be applied to interconnect business processes by outsourcing and composing process services.

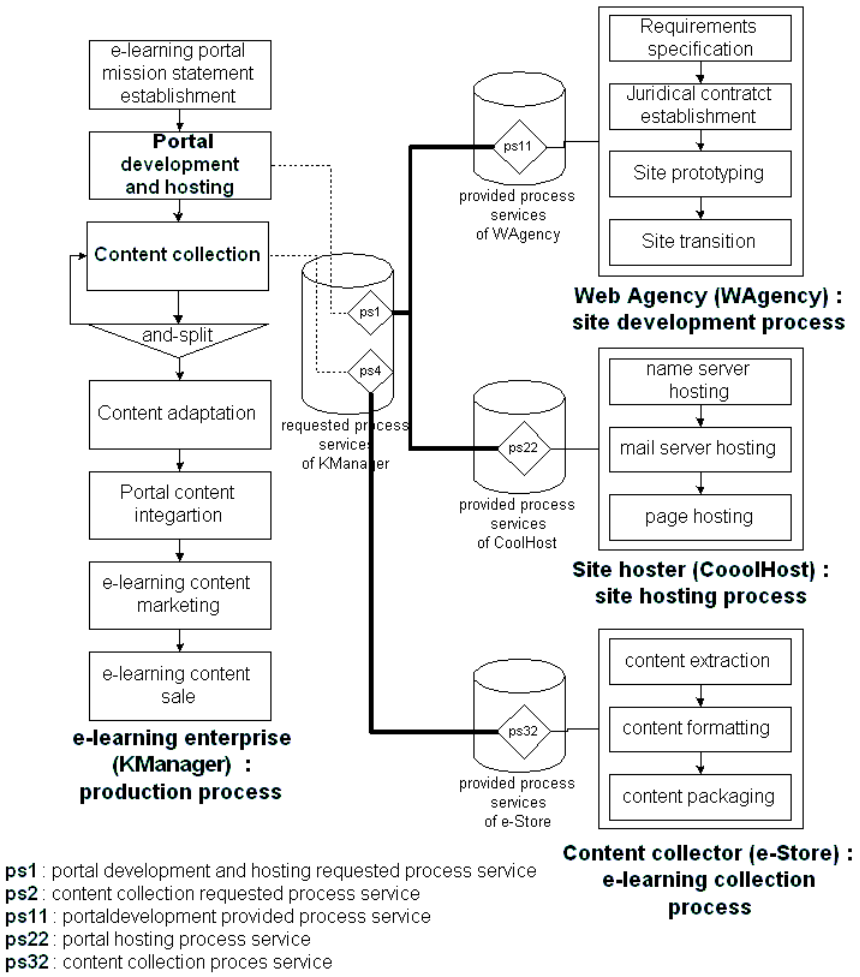

Fig. 1. An interconnection example of an e-learning enterprise process services

Within this example, we consider the following set of process services PS, set of data D, set of access rights AR, set of methods M, and set of events E:

- $PS = PS_{prv} \cup PS_{req}$: the union of provided and requested process services where
  - $PS_{prv} = \{ps_{11}$ (portal development process service), ps_{22} (portal hosting process service), ps_{32} (content collection process service)$\}$,
  - $PS_{req} = \{ps_1 := (ps_{11}, ps_{22})$ (composite service), $ps_4 := ps_{32}$ (atomic process service)$\}$;
- $D = \{d_1$: portal development judicial contract, d_2: portal hosting judicial contract, d_3: graphical charter of the portal development, d_4: portal prototype. . . $\}$;
- $AR = \{$read, write$\}$;
- $M = M_{ps_{11}} \cup M_{ps_{22}} \cup M_{ps_{32}}$ where
  - $M_{ps_{11}} = \{$getPortalDevelopmentAllocatedHumanResources(),. . . $\}$,
  - $M_{ps_{22}} = \{$getPortalHostingStatus(),. . . $\}$,
  - $M_{ps_{32}} = \{$getProcessInstanceAchievedModuleDuration(),. . . $\}$
- $E = E_{ps_{11}} \cup E_{ps_{22}} \cup E_{ps_{32}}$ where
  - $E_{ps_{11}} = \{$ASK_FOR_MORE_SPECIFICATION,. . . $\}$,
  - $E_{ps_{22}} = \{$BEGIN_PORTAL_HOSTING. . . $\}$,
  - $E_{ps_{32}} = \{$END_MODULE_COLLECTION. . . $\}$.

According to the approach we have presented above, we aim to interconnect processes through process service interaction. We describe interaction through two dimensions: data sharing and process service coordination. We present, in the following, these features that we will integrate into our model for process service interaction.

3 Process Service Information Sharing

The behavior of process service interactions depends on the nature of operations they perform on the shared information. Process services can share common data and/or access to private data using methods published by their process service owners. These accesses have to be controlled so that every process service provider plays its real role in the composition. For this reason, we define access contracts for both shared and private data.

We define an access contract as the association of a service, an access right, and an object. We denote an access contract by *Access*.
$Access(ps, o, ar) \stackrel{\text{def}}{=}$ the process service ps has the access right ar on the object o modelling some. We express access contracts for shared data in the example presented in section 2 as:

- $Access(ps_1, d_1, write)$ (the composite service for portal hosting and development can write the portal development contract),
- $Access(ps_{22}, d_3, read)$ (the provided service for portal hosting can read graphical charter of the portal development).

In a composite process service, each process service provides a set of access points to its resources for other cooperating process services. These points include methods remote process services can execute in order to access to private data. The access to these methods is controlled by execution rights:

$Exec(ps_i, M^{'}) \stackrel{\text{def}}{=} \forall m_{ps_j} \in M^{'}$, the process service ps_i can execute the method m of the process service ps_j $(i \neq j)$.

In the example presented in section 2 methods of the provided services ps_{11} and ps_{22}, the required process services ps_1 and ps_4 are $Exec(ps_1, M^{'}_{ps_{11}} \cup M^{'}_{ps_{22}})$ and $Exec(ps_4, M^{'}_{ps_{32}})$ where

- $M^{'}_{ps_{11}} = \{$getPortalDevelopmentAllocatedHumanResources(), getPortalDevelopmentStatus(), checkinDocument(), checkoutDocument()$\}$ (NB: $M^{'}_{ps_{11}} \subset M_{ps_{11}}$),
- $M^{'}_{ps_{22}} = \{$getPortalHostingStatus(), checkinDocument(), checkoutDocument()$\}$ (NB: $M^{'}_{ps_{22}} \subset M_{ps_{22}}$),
- $M^{'}_{ps_{32}} = \{$getProcessInstanceAchievedModuleDuration(), checkinDocument(), checkoutDocument()$\}$ (NB: $M^{'}_{ps_{32}} \subset M_{ps_{32}}$).

In addition to method visibility, it is necessary to provide to cooperating process services means to coordinate their actions in an informal way and to work in an autonomous way while being aware of what occurs in the composite process service. For this reason, each process service has to provide events allowing other services to know the state of its execution and its private data. Since each process service is interested in a specific information, it is beneficial to control the visibility of process service events:
$Recept(ps_i, E') \stackrel{\text{def}}{=} \forall e_{s_j} \in E'$, the process service ps_i can receive the visible event e of the process service ps_j $(i \neq j)$.

Concerning event visibility, the process services ps_1 and ps_4 can receive from the process services ps_{11}, ps_{22} and ps_{32} the following sets of events : $Recept(ps_1, E^{'}_{ps_{11}} \cup E^{'}_{ps_{22}})$ and $Recept(ps_4, E^{'}_{ps_{32}})$ where

- $E^{'}_{ps_{11}} = \{$ASK_FOR_MORE_SPECIFICATION,ENDPORTAL_DEVELOPMENT$\}$ (NB: $E^{'}_{ps_{11}} \subset E_{ps_{11}}$),
- $E^{'}_{ps_{22}} = \{$BEGIN_PORTAL_HOSTING,ASKFORMORE_SPECIFICATION$\}$ (NB: $E^{'}_{ps_{22}} \subset E_{ps_{22}}$),
- $E^{'}_{ps_{32}} = \{$END_MODULE_COLLECTION, PRODUCED_NEW_MODULE_VERSION$\}$ (NB: $E^{'}_{ps_{32}} \subset E_{ps_{32}}$).

4 Process Service Coordination

We define process service coordination as a constrained interaction (data sharing, event reception, and method invocation). This coordination can be based on their states or on the states of their shared data.

4.1 Coordination Based on Process Service States

We define a coordination contract based on process service states as a set of coordination rules. These rules denote an association of two process services and a type of coordination. Each coordination type defines a constraint on the states of both process services. In order to give some coordination rule examples, we introduce the concepts of process service states, state sequences, and process service view sequence. Each process service has a state which reflects its situation or circumstances that describe it at a given time. We call a state sequence of a process service ps, that we denote S^{ps}, the ordered complete set of states of ps. We can define S^{ps} as $S^{ps} = \{s_0^{ps}, s_1^{ps}, \dots\}$, where s_0^{ps} is the initial state of ps and s_1^{ps}, s_2^{ps},... some of its successive states. A state of a process service ps_1 is observed by a process service ps_2 if this latter has executed a method of ps_1 that returns a value reflecting a state of ps_1, or if ps_2 receives an event from ps_1 showing its state.

For a process service ps_1, we call a view sequence of process service ps_2, noted $S_{ps_2}^{ps_1}$, the subset of S^{ps_1} (states of ps_1) observed by ps_2. Considering a process service with a begin and an end, the set of methods it can execute and the set of events it can receive are finite. Thus, the set $S_{ps_2}^{ps_1}$ is finite. Let $S_{ps_2}^{ps_1} = \{s_{ps_2,0}^{ps_1}, s_{ps_2,1}^{ps_1}, \dots, s_{ps_2,n-1}^{ps_1}\}$ be the state sequence of the process service ps_1 observed by ps_2, where n is the number of ps_1 states observed by ps_2. $s_{ps_2,0}^{ps_1}$ is the initial state of ps_1 observed by ps_2 after its beginning and $s_{ps_2,n-1}^{ps_1}$ is the last state of ps_1 observed by ps_2 before its end. Note that there may be some states of ps_1 not observed by ps_2 and which can be produced between two states of ps_1 observed by ps_2. We call view sequence of a process service ps all its states and all the states of other process services that it can observe. Let $V^{ps} = S^{ps} \cup_{ps_1 \in PS} S_{ps}^{ps_1}$ be the view sequence of the process service ps. This set represents the states of the process service that ps is aware about.

Let $SCoor(ps_1, ps_2, coortype)$ be a coordination rule based on process service states, where ps_1 and ps_2 are two process services from the process service set PS and $coortype$ be a coordination type belonging to the state based coordination type set $SCoorT$. Coordination types we present are inspired from several works. We mention, among others, those achieved in the transactional field [7], and those accomplished in workflow interconnection field [24]. In order to introduce these coordination types, we use the following notations. Let ps be a composite process service, ps_i and ps_j two of its process service components. Let S^{ps_i} and S^{ps_j} be their respective state sequences. Let S (s state sequence) be the union of S^{ps_i} and S^{ps_j}. We supply S by a partial order relation we denote $\rightarrow$ that handles the causality of S elements.

Considering a process service as a transaction with a begin (enact), an end (commit, abort), and a life cycle, we can express dependencies on process service states using the transactional approach presented in [7]. Process service state significant events include Enacted, Preempted, Committed, Aborted,....

Here are some types of state based process service coordination for describing:

– serial dependency of two process services (*ECD*: Enact-On-Commit Dependency):

$$SCoor(ps_2, ps_1, ECD) \stackrel{\text{def}}{=} \mathsf{Enacted}_{ps_2} \in S \Rightarrow \mathsf{Committed}_{ps_1} \rightarrow \mathsf{Enacted}_{ps_2}$$

– commit dependency of two process services (*CD*: Commit Dependency):

$$SCoor(ps_2,\ ps_1,\ CD) \stackrel{\text{def}}{=} \mathsf{Committed}_{ps_2} \in S \Rightarrow (\mathsf{Committed}_{ps_1} \in S \Rightarrow (\mathsf{Committed}_{ps_1} \rightarrow \mathsf{Committed}_{ps_2}))$$

Interface 4 of the WfMC (Workflow Management Coalition) [24] supports workflow processes interconnection by proposing paradigms for workflow interoperability. Based on the approach of the WfMC paradigms, we define two new process service state based coordination types describing:

– asynchronous externalisation of a sub process service (*CMD*: Chained Model Dependency):

$$SCoor(ps_2,\ ps_1,\ CMD) \stackrel{\text{def}}{=} \mathsf{Enacted}_{ps_2} \in S \Rightarrow \mathsf{Enacted}_{\ ps_1} \rightarrow \mathsf{Enacted}_{ps_2}$$

– synchronous externalisation of a sub process service (*NMD*: Nested sub-process Model Dependency)

$$SCoor(ps_2, ps_1, NMD) \stackrel{\text{def}}{=} (\mathsf{Enacted}_{ps_2} \in S \Rightarrow \mathsf{Enacted}_{ps_1} \rightarrow \mathsf{Enacted}_{ps_2}) \wedge (\mathsf{Committed}_{ps_1} \in S \Rightarrow \mathsf{Committed}_{ps_2} \rightarrow \mathsf{Committed}_{ps_1})$$

These types of process service state based coordination have been given as simple examples. We have developed other coordination types that we do not present in this paper[1].

The following state based coordination contracts are expressed for the process service of the section 2 :
$SCoor(ps_{11},\ ps_1,\ NMD)$: the process service ps_{11} (portal development provided process service) is a nested sub-process of the process service ps_1 (portal development and hosting requested process service) and $SCoor(ps_{22},\ ps_1,\ NMD)$: the process service ps_{22} (portal hosting provided process service) is a nested sub-process of the process service ps_1.

4.2 Coordination Based on Data States

We define a coordination contract based on data states as the association of two process services, an object (modeling the data), and a type of coordination ([20], [21]). To define formally coordination contracts, we use the notion of object state

[1] An example of coordination types we have developed is Parallel Synchronized Model Dependency: ps_1 and ps_2 are parallel synchronized if they agree on a synchronization point within which they exchange data then they continue their executions normally.

sequence and process service view sequence that we define in the following before presenting coordination contracts based on data states.

We call the state sequence of an object o, denoted S^o, the ordered set of the states of o. We can define S^o as $S^o = \{s_0^o,\ s_1^o, \ldots\}$, where s_0^o is the initial state of the object and $s_1^o,\ s_2^o,\ \ldots$ are the successive states of o. An object can change its state when a process service performs an operation that changes the value of one of its attributes. A process service can observe a state of the object if it performs an operation which returns a value reflecting the state of this object. Note that no assumptions are made on the granularity of a state change. Depending on the context and of the application being specified, this may vary from one single character change in a textual document to a CAD [2] object replacement. We call the view sequence of a process service ps on an object o, noted S_{ps}^o, the ordered subset of S^o (the state sequence of o) restricted to the states observed by ps.

Given an object, a coordination contract binds two process services to express the fact that there is a constraint on their view sequences. The coordination is a *process service coordination* if it expresses constraints on the sequence of states observed by one or the other of the both process services. It is a *state coordination*, if it expresses constraints on only one state observed by one or the other of the both process services.

For a given object o, the process service ps_1 is coordinated to the process service ps_2 on o when $S_{ps_1}^o$ is included in $s_{ps_2}^o$. We note this coordination type *SvCoor* (Process Service Coordination).

$$DCoor(ps_1, ps_2, o, SvCoor) \stackrel{\text{def}}{=} S_{ps_1}^o \subset S_{ps_2}^o.$$

ps_1 and ps_2 are state coordinated if there exists a state $s_{ps_1,i}^o$ observed by ps_1 and a state $s_{ps_2,j}^o$ observed by ps_2 that are identical. We note this coordination type *StateCoor* (State Coordination).

$$DCoor(ps_1,\ ps_2,\ o,\ StateCoor) \stackrel{\text{def}}{=}$$
$$\exists\ i \in \{0..n-1\},\ \exists j \in \{0..m-1\}\ s_{ps_1,i}^o = s_{ps_2,j}^o$$

where n (respectively m) denotes the number of states observed by ps_1 (respectively ps_2).

This type of coordination can be used to specify the more general case of two process services coordinated on an arbitrary state. However, there is a lot of situations in which such coordination occurs at the beginning or at the end of one or both of the coordinated process services.

For this reason, we provide specialized coordination types corresponding to the coordination of :

- a process service initial state and another process service state (Initial state Coordination):

[2] Computer Aided Design.

$$DCoor(ps_1, ps_2, o, ICoor) \stackrel{\text{def}}{=} s^o_{ps_1,0} \in S^o_{ps_2}$$

– the initial states of two process services (Initial States Coordination):

$$DCoor(ps_1, ps_2, o, IsCoor) \stackrel{\text{def}}{=} s^o_{ps_1,0} = s^o_{ps_2,0}$$

– a process service final state and another process service state (Final state Coordination):

$$DCoor(ps_1, ps_2, o, FCoor) \stackrel{\text{def}}{=} s^o_{ps_1,n-1} \in S^o_{ps_2},$$
where n is the number of the states of o observed by ps_1

– the final states of two process services (Final States Coordination):

$$DCoor(ps_1, ps_2, o, FsCoor) \stackrel{\text{def}}{=} s^o_{ps_1,n-1} = s^o_{ps_2,m-1}$$
where n (respectively m) is the number of the states of o observed by ps_1 (respectively ps_2)

– a process service final state with another process service initial state (Serial Coordination):

$$DCoor(ps_1, ps_2, o, SrlCoor) \stackrel{\text{def}}{=} s^o_{ps_1,n-1} = s^o_{ps_2,0}$$

– two state sequences of two process services (Total Coordination):

$$DCoor(ps_1, ps_2, o, TotCoor) \stackrel{\text{def}}{=} n = m \wedge \forall\ i\ \in\ \{0..n-1\}, s^o_{ps_1,i} = s^o_{ps_2,i}$$

There are some dependencies between the different coordination types that lead to implicit coordination types when we define some other stronger coordination type. For instance we have the following dependencies:

– if the process service ps_1 is coordinated to the process service ps_2, then the ps_1 initial state is coordinated to another ps_2 state:

$$DCoor(ps_1, ps_2, o, SvCoor) \Rightarrow DCoor(ps_1, ps_2, o, ICoor)$$

– if the initial states of ps_1 and ps_2 are coordinated, then the ps_1 initial state is coordinated to another ps_2 state and the ps_2 initial state is coordinated to another ps_1 state:

$$DCoor(ps_1, ps_2, o, IsCoor) \Rightarrow$$
$$DCoor(ps_1, ps_2, o, ICoor) \wedge DCoor(ps_2, ps_1, o, ICoor)$$

– if the process service ps_1 is coordinated to the process service ps_2, then the ps_1 final state is coordinated to another ps_2 state:

$$DCoor(ps_1, ps_2, o, SvCoor) \Rightarrow DCoor(ps_1, ps_2, o, FCoor)$$

– if the final states of ps_1 and ps_2 are coordinated, then the ps_1 final state is coordinated to another ps_2 state and the ps_2 final state is coordinated to another ps_1 state:

$$DCoor(ps_1, ps_2, o, FsCoor) \Rightarrow DCoor(ps_1, ps_2, o, FCoor) \wedge DCoor(ps_2, ps_1, o, FCoor)$$

– we quote, in addition, here that the total coordination is a strong coordination type that implicit implies other ones. If ps_1 and ps_2 are totally coordinated, then
  - ps_1 is coordinated to ps_2 and ps_2 is coordinated to ps_1:

$$DCoor(ps_1, ps_2, o, TotCoor) \Rightarrow DCoor(ps_1, ps_2, o, SvCoor) \wedge DCoor(ps_2, ps_1, o, SvCoor)$$

  - the final states of ps_1 and ps_2 are coordinated:

$$DCoor(ps_1, ps_2, o, TotCoor) \Rightarrow DCoor(ps_1, ps_2, o, FsCoor)$$

  - the ps_1 final state is coordinated to another ps_2 state and the ps_2 final state is coordinated to another ps_1 state:

$$DCoor(ps_1, ps_2, o, TotCoor) \Rightarrow DCoor(ps_1, ps_2, o, FCoor) \wedge DCoor(ps_2, ps_1, o, FCoor)$$

  - the initial states of ps_1 and ps_2 are coordinated:

$$DCoor(ps_1, ps_2, o, TotCoor) \Rightarrow DCoor(ps_1, ps_2, o, IsCoor)$$

  - the ps_1 initial state is coordinated to another ps_2 state and the ps_2 initial state is coordinated to another ps_1 state:

$$DCoor(ps_1, ps_2, o, TotCoor) \Rightarrow DCoor(ps_1, ps_2, o, ICoor) \wedge DCoor(ps_2, ps_1, o, ICoor)$$

In the example presented in section 2 the coordination contracts $DCoor(ps_1, ps_{11}, d_1, TotCoor)$ based on the data states of the process services shows that the required process service for portal development and hosting ps_1, and the provided process service for portal development ps_{11} are totally coordinated on the portal development contract d_1.

5 Process Service Interaction Model

In this section, we propose an interaction model which supports information sharing and coordination between cooperating process services. Thus, this model can describe needs of roles and coordination management for cooperating process services.

Formally, process service interaction model is represented by the tuple (PS, D, CL) where PS is the set of process services, D is the set of shared documents, and CL (Contract List) is a set of cooperation contracts. Each contract binds a set of process services participating to a cooperation by:

- DAL (Data Access List), a set of dynamic access rights on shared documents;
- MAL (Method Access List), a set of dynamic execution rights on methods made accessible by process services;
- EAL (Event Access List), a set of dynamic reception rights on events produced by process services;
- DCL (Data Coordination List), a set of dynamic coordination rules based on process service shared information state;
- SCL (Service Coordination List), a set of dynamic coordination rules based on process service states.

By dynamic, we mean negotiation based customisation [14].

Here is an example of the tuple (PS, D, CL):

$PS = PS_{frn} \cup PS_{req}$: the union of the following provided and requested process services $PS_{frn} = \{ps_{11}$: portal development process service, ps_{22}: portal hosting process service, ps_{32}: content collection process service$\}$ and $PS_{req} = \{ps_1 := (ps_{11}, ps_{22})$ (composed process service), $ps_4 := ps_{32}$ (atomic process service)$\}$

$D = \{ d_1$: portal development judicial contract, d_2: portal hosting judicial contract, d_3: portal development graphical charter, d_4: portal prototype, d_5 : e-learning module requirements,$\dots\}$

and CL:

$DAL = \{Access(ps_1, d_1, read), Access(ps_1, d_1, write), Access(ps_{22}, d_3, read), ...\}$,
$MAL = \{Exec(ps_1, M'_{ps_{11}} \cup M'_{ps_{22}}), Exec(ps_4, M'_{ps_{32}}), ...\}$,
$EAL = \{Recept(ps_1, E'_{ps_{11}} \cup E'_{ps_{22}}), Recept(ps_4, E'_{ps_{32}}), ...\}$,
$DCL = \{DCoor(ps_1, ps_{11}, d_1, TotCoor), DCoor(ps_{11}, ps_{22}, d_3, IsCoor), ...\}$,
$SCL = \{SCoor(ps_{11}, ps_1, NMD), SCoor(ps_{22}, ps_1, NMD), ...\}$.

with

$M'_{ps_{11}}$ = {getPortalDevelopmentAllocatedHumanResources(), getPortalDevelopmentStatus(), checkinDocument(), checkoutDocument()},
$M'_{ps_{22}}$ = {getPortalHostingStatus(), checkinDocument(), checkoutDocument()},
$M'_{ps_{32}}$ = {getProcessInstanceAchievedModuleDuration(), checkinDocument(), checkoutDocument()},
$E'_{ps_{11}}$ = {ASK_FOR_MORE_SPECIFICATION,END_PORTAL_DEVELOPMENT},
$E'_{ps_{22}}$ = {BEGIN_PORTAL_HOSTING,ASK_FOR_MORE_SPECIFICATION}, and
$E'_{ps_{32}}$ = {END_MODULE_COLLECTION,PRODUCED_NEW_MODULE_VERSION}

Cooperation rules define explicitly allowed, refused, and compulsory operations. Information sharing rules, we may define inside a cooperation contract,

assign (negative, or positive) rights on shared documents, on accessible events and/or methods. Each positive access right enables its possessing process service to execute a set of operations on shared or private documents, or to receive events reflecting states of data of other process service. Negative information sharing rules prohibit to a process service to execute a set of operations or the reception of a set of events. To respect certain coordination rules inside a cooperation contract, the participating process services have to observe the shared information states. Actually, if it exists a coordination rule of $DCoor(ps_1, ps_2, d, coortype)$ type, then ps have to observe at least a state of d when $coortype$ belongs to $\{IsCoor, FsCoor, TotCoor, SrlCoor\}$.

For a process service ps that participate to a cooperation contracts, we can define consequently a set of operations that ps can execute, a set of operations that ps cannot execute, and a set of operations that ps ought to execute. Process service interaction model defines cooperation contracts between process services. These contracts model rules for process interconnection. However, cooperation consistency has to be maintained in order to avoid contradiction between agreed contracts. A process service ps can respect a cooperation contract if every compulsory operations are accessible, and every compulsory operations are not prohibited. A cooperation contract is thus coherent if it can be respected by every cooperating process services.

6 Conclusion

Our paper is a contribution in the field of process interconnection. Our process service interaction model allows communication and coordination between process services aiming to interconnect processes, and, therefore, to support inter-enterprises cooperation. Actually, our process service interaction model encompasses information sharing between process services, and process services coordination. Therefore our model allows management of roles and coordination of services through access rights on shared data, execution rights on process services methods, reception rights on process services events, and coordination rights based on data states or on process service states.

Although implementation is not in the intended scope of this paper, we can mention that we are currently developing an implementation of our process service interaction model within *DISCOBOLE* (DIStributed COoperation and Business prOcess on LinE), our cooperative platform for process service interconnection. *DISCOBOLE* is developed in Java on a CORBA bus. Our implementation relies on a process service structure which may be seen as the development of a design pattern for cooperation. Actually, a process service may be related to proxy and adapter patterns. While the proxy pattern limits accesses to process resources (data access rights $Access(ps, d, dr)$, method visibility $Exec(ps,\ M)$, visibility of process service events $Recept(ps,\ E)$), the adapter pattern provides a new interface to the adapted process service (ability to exchange data, coordination ability, ability to control interaction consistency).

Our next step will be fully explore the cooperation pattern paradigm we have used to implement our interaction service model. In fact, our model description and our implementation of it with patterns present two orthogonal views of process service interaction. We plan to cross fertilize and validate formally these two orthogonal views to produce an integrated and complete view.

References

1. Baïna, K., Benali, K. and Godart, C.: A process service model for dynamic enterprise process interconnection. In *9th Int. Conf. on Cooperative Information Systems, In Cooperation with VLDB 2001*, volume 2172 of *LNCS*, pages 239–254, Trento, Italy, September 5–7, 2001. Springer-Verlag.
2. K. Baïna, F. Charoy, C. Godart, D. Grigori, S. el Hadri, H. Skaf, S. Akifuji, T. Sakaguchi, Y. Seki, and M. Yoshioka. CORVETTE: A Cooperative Workflow Development Experiment. In L. M. Camarinha-Matos, editor, *3rd IFIP Working Conference on Collaborative Business Ecosystems and Virtual Enterprises (PRO-VE'02)*, pages 169–180, Sesimbra, Portugal, May 1–3, 2002. Kluwer Academic Publishers.
3. A. P. Barros and A. H. M. Ter Hofstede. Modelling extensions for concurrent workflow coordination. In *4th IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'99)*, pages 336–347, Edinburgh, Scotland, September 2–4, 1999. IEEE Computer Society Press.
4. G. A. Bolcer and G. Kaiser. Swap: Leveraging the web to manage workflow (swap). In *IEEE Internet Computing*. IEEE Computer Society, http://computer.org/internet/, January-February 1999.
5. B. Benatallah, B. Medjahed, A. Boughettaya, A. Elmagarmid, and J. Beard. Composing and maintaining web-based virtual enterprises. In *1st Workshop on Technologies for E-Services, In cooperation with VLDB 2000 (TES'00)*, Cairo, Egypt, September 14–15, 2000.
6. F. Casati, S. Ilnicki, L. J. Jin, and M. C. Shan. eflow: an open, flexible, and configurable approach to service composition. In *2nd International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS'00)*, pages 125–132, Milpitas, California, June 8–9, 2000.
7. P-K. Chrysanthis and K. Ramamritham. ACTA: The SAGA continues. In *Database Transaction Models for Advanced Applications*, pages 349–397. 1992.
8. K. Dutta, D. VanderMeer, A. Datta, and K. Ramamritham. User Action Recovery in Internet SAGAs (iSAGAs). In F. Casati, D. Georgakopoulos, and M-C. Shan, editors, *2nd Workshop on Technologies for E-Services, In Cooperation with VLDB'2001 (TES'01)*, number 2193, pages 132–146, Rome, Italy, September 14–15, 2001. Springer-Verlag.
9. M.-C. Fauvet, M. Dumas, B. Benatallah, and H. Paik. Peer-to-peer traced execution of composite services. In F. Casati, D. Georgakopoulos, and M-C. Shan, editors, *2nd Workshop on Technologies for E-Services, In cooperation with VLDB 2001 (TES'01)*, volume 2193 of *LNCS*, pages 103–117, Rome, Italy, September 14–15, 2001. Springer-Verlag.
10. D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker. Managing process and service fusion in virtual enterprises. *Information Systems, Special Issue on Information Systems Support for Electronic Commerce*, 24(6), 1999.
11. C. Hagen and G. Alonso. Beyond the black box: Event-based inter-process communication in process support systems. In *19th International Conference on Distributed Computing Systems (ICDCS'99)*, Austin, Texas, USA, May/June 1999.

12. IBM. *IBM open sources technology for accessing UDDI Business Registry.* www-3.ibm.com/services/uddi/announce012501.html, January 2001.
13. IONA. *Orbix E2A, Web Services Integration Platfom.* IONA, www.iona.com, 2002.
14. M. Munier, K. Baïna, and K. Benali. A negotiation model for CSCW. In O. Etzion and P. Scheuermann, editors, *5th IFCIS Int. Conf. on Cooperative Information Systems, In Cooperation with VLDB 2000 (CoopIS'00)*, volume 1901 of *LNCS*, pages 224–235, Eilat, Israel, September 6–8, 2000. Springer-Verlag.
15. Microsoft. *Microsoft .NET.* www.microsoft.com/net/, 2001.
16. OMG. Workflow Management Facility Convenience Document combining dtc/99-07-05 dtc/2000-02-03 (WF RTF 1.3 Report). *OMG, ww.omg.org*, February 2000.
17. M. Reichert and P. Dadam. A framework for dynamic changes in workflow management systems. In *8th International Workshop on Database and Expert Systems Applications (DEXA'97)*, Toulouse, France, September 1–2, 1997.
18. M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In Won Kim, editor, *Modern Database Systems, The Object Model Interoperability and beyond*, pages 592–620. Addison Wesley, ACM Press, 1995.
19. Sun. Sun Open Net Environment (Sun ONE). *Sun microsystems, www.sun.com/sunone/*, 2002.
20. S. Tata. Outils pour la description et la mise en œuvre des interactions coopératives dans les équipes virtuelles. *Thèse en informatique*, Université Henry Poincaré – Nancy I, Loria, Décembre 2000.
21. S. Tata. Policies for Cooperative Virtual Teams. In *Proceedings of the 5th International Conference on Coordination Models and Languages* COORDINATION 2002, York, UK, April 8–11, 2002.
22. UDDI.Org. UDDI V. 2.0 API Specification. *www.uddi.org*, June 8, 2001.
23. W3C. Simple Object Access Protocol (SOAP) V. 1.1. *W3C (World Wide Web Consortium), www.w3.org/TR/SOAP/*, 2000.
24. WFMC. Workflow Standard – Interoperability, Abstract Specification, WFMC-TC-1012, V. 1.0. *Workflow Management Coalition, www.wfmc.org*, October 20, 1996.

Exception Handling in the BPEL4WS Language

Francisco Curbera[1], Rania Khalaf[1], Frank Leymann[2], and Sanjiva Weerawarana[1]

[1] IBM TJ Watson Research Center, Hawthorne, NY 10532, USA
{curbera,rkhalaf,sanjiva}@us.ibm.com
[2] IBM Software Group, Boeblingen, Germany
ley1@de.ibm.com

Abstract. Graph oriented models are at the core of most business process management systems. In recent years, "algebraic" business process modeling languages based on different process calculi have been proposed. The semantics of these algebraic process languages are quite different, and seemingly incompatible, with those of graph oriented approaches. In this paper we study how the BPEL4WS exception handling mechanism is used to integrate the algebraic and graph process models. Unlike other approaches to exception handling in business processes, the BPEL4WS model does not require that the process topology be constrained by the exception handling hierarchy, thus allowing both highly structured and graph based processes to benefit from it. Based on this exception handling model, we explain "dead path elimination" (the runtime mechanism by which process termination is ensured) as a form of exception processing. The integration of dead path elimination with the exception handling mechanism provides the semantic base for the integration of the graph and algebraic processes models in BPEL4WS.

Subject Classifications: Business process modeling, reference models, process patterns, workflow management systems.

1 Introduction

Graph oriented models [3,14] have been at the core of business process management systems for over two decades. Most of the industry and a good part of the academic community approach business processes assuming a graph oriented model, in one shape or another. In recent years several proposals have been made to base business process languages and systems on process calculi such as Milner's Pi calculus [15] and Fournet's join calculus [9]. These "algebraic" approaches have lead to a new perspective in business process design that radically departs from the graph-oriented model and results in highly structured modeling languages [17,2].

The critical design issue of the BPEL4WS [6] language has been whether it is possible to integrate the graph oriented and algebraic approaches within a unified semantic framework. From a practical perspective, incorporating the

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 276–290, 2003.

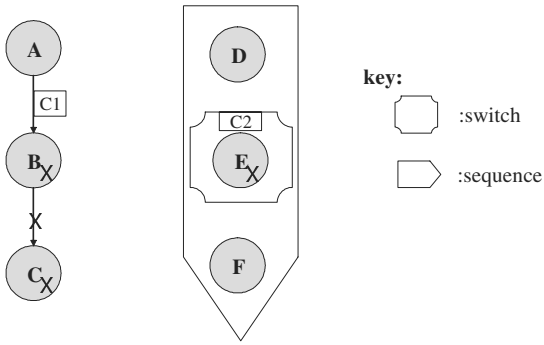

Fig. 1. Different operational semantics for graph oriented and algebraic process models.

structured control constructs of algebraic languages into a graph based model can yield many advantages. Structured exception handling, for example, helps deal with process errors in a consistent manner and results in clearer and easier to manage processes. Structured control artifacts, such as while loops, provide simple ways of encoding predefined, complex behavior, ensuring correctness. This paper explains how BPEL4WS incorporates the algebraic modeling approach with a graph oriented model.

BPEL4WS, the Business Process Execution Language for Web Services, assumes the graph metamodel of WSFL [7], based on a particular derivation of Petri nets [14], and the algebraic metamodel of Xlang [17], which derives from the Pi calculus [15]. The most important difference between these lies in their distinct operational semantics. In the graph oriented semantics activities are automatically skipped (or disabled) when they cannot be performed, due for example to a boolean condition evaluating to false at a certain point in the graph. This technique is known as *dead path elimination* (DPE), and is described in Section 2 together with the graph and structured process metamodels. DPE propagates the disabled state transitively across the process graph; as a result, the local evaluation of a boolean condition to false can have global effects in the process. In the algebraic process model, control constructs are nested and the effects of a local evaluation are strictly scoped. Consider the examples in Figure 1. On the graph-oriented process on the left, when condition C1 evaluates to false activities B and C are not executed due to DPE propagation. For the process on the right a false value of condition C2 only implies not executing E; however, F is not affected and executes normally. If we naively integrated both parts of the figure by adding a connector linking B to E, it would not be clear how or whether DPE should be applied.

The goal of this paper is to explain how this problem is solved by BPEL4WS. The answer is provided in two steps. First, structured exception handling is incorporated into BPEL4WS in a way such that it works with both structured processes and arbitrary graphs. Then, a new system exception is introduced to represent the event by which an activity or task is disabled. The result is that

we are able to explain the operational semantics of both DPE and structured constructs in terms of a common exception handling mechanism.

This rest of this paper is organized as follows. In Section 2 we provide an overview the basic operational semantics on which BPEL4WS is based, including basic process graphs and algebraic constructs. Section 3 discusses prior work on incorporating exception handling in workflows, focusing on their requirements on the process structure. Section 4 describes the BPEL4WS exception model. Section 5 explains how BPEL4WS explains DPE by casting it as a form of exception handling, and how this leads to the seamless integration of graphs and algebraic control primitives. Finally, in sections 6 and 7 we review future work and summarize the contributions of this paper.

2 Process Metamodel

In established workflow systems [1],[14], [10], a workflow consists of a set of individual activities or tasks, each defining a piece of business logic. Activities are linked into a directed graph using *control connectors* (links) that constrain the order of execution of tasks; in WSFL, but not in BPEL4WS, *data connectors* are used to define the transfer of data between them. Initially, all activities with no control dependencies are activated. When an activity completes its execution, it activates the control links that originate from it. The positive or negative activation of each link is governed by a *transition condition*, a boolean expression relating the values of data fields and other state of the process. A *join condition* is a boolean expression in terms of the values of incoming control links, which determines whether or not an activity is activated for execution. A join condition also serves as a synchronization point in the process, since it forces the execution of an activity to wait until all incoming parallel execution paths reach the join.

The corresponding simplified activity lifecycle diagram is shown in Figure 2. Initially, an activity is in the *default* state. Once the values of all its incoming links are known, the join condition (shown in the diagram as "JC") is evaluated in order to decide whether or not it should run. If the join condition evaluates to true, the activity enters the *activated* state and runs until it completes. Otherwise, the activity does not run and goes into the *disabled* state. The "revive" arc is included to enable iterative execution of a particular activity.

Dead path elimination [13] is the process by which the disabled state of an activity is propagated downstream through the process graph. Activities that are disabled set the value of their outgoing control links to false. The join conditions of all target activities are then evaluated (assuming the values of all incoming control links are available) and the activities are disabled if the condition evaluates to false. The process continues downstream until no more join conditions can be evaluated. DPE ensures that all wait states introduced by join conditions are eventually terminated, and thus provides first class support for the common "synchronizing merge" pattern, see [18], in which the workflow must wait for multiple paths to converge.

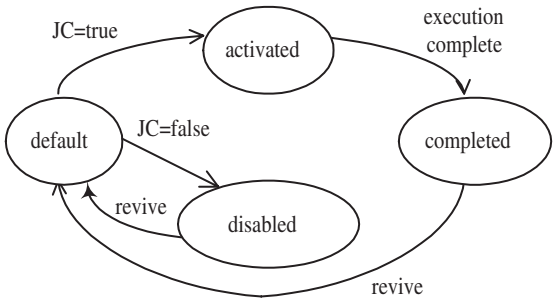

Fig. 2. Simplified activity lifecycle diagram.

DPE is implicitly performed by the process engine. Under the assumption that the execution of individual activities eventually terminates, and that process graphs are acyclic, DPE allows a process engine to decide whether a process instance has terminated by checking the state of its activities. A process terminates when every activity has entered the completed or disabled state (additional tests are required if iterative execution is enabled).

BPEL4WS does not use data links. Instead, data containers are global and may be accessed by any activity in the process. Activities specify in their definition which data container they will read from upon beginning and which container they will write to upon completion.

2.1 Algebraic Control Constructs in BPEL4WS

The BPEL4WS process model presents a number of enhancements to the basic model introduced above. One of the main characteristics is BPEL4WS's merging of the algebraic authoring style with the graph-oriented style of process modeling.

In the algebraic style of authoring, such as the one exhibited by Xlang [17], there are two kinds of activities, elementary and structured. Elementary activities perform basic actions, while structured activities act as strictly nested blocks that contain other activities and impose execution semantics on them.

BPEL4WS processes are created by aggregating elementary and structured activities using both the control constructs provided by the structured activities and control links to connect activities (of any kind) into execution graphs. Elementary activities include the execution of an operation of a business component, a data manipulation step, the need to wait for a set delay, and the raising of an exception. There are several types of structured activities, providing several basic execution models: ordered sequence of steps (the *sequence* activity), branching using the now common "case statement" approach (the *switch* activity), iterative execution (the *while* activity), execution of one of several alternative paths based on an incoming event (the *pick* activity), and also a basic variant of a process flow (the *flow* activity, which can contain an execution graph). Control links

must be contained within a flow activity and require that the source and the target of the link also be contained in the flow (at any level of nesting). Note that links may cross the boundaries of structured activities, except for while loops. Of special importance is the *scope* construct which provides a unit of recovery and exception handling in BPEL4WS and which will be discussed in more detail in Section 4.

To simplify the diagrams and clarify the discussion, in the the rest of this paper we will make the simplifying assumption that all scopes as well as the process itself are equipped with a nested flow activity. This allows us to define arbitrary graphs inside a scope with minimal graphical notation.

3 Exception Handling and Process Structure

Exceptions have been part of the business process modeling for over 20 years now, dating back to the work of Eder and Liebhart [8] and Reuter [16], and are supported by most commercial workflow products. Structured exception handling, on the other hand, is a common feature in many structured programming languages such as C++ and Java. Structured exception handling mechanisms associate exception handlers with specific program blocks. Handlers factor out common error handling code and cleanly separate the "main" and "alternate" execution paths. By properly nesting exception handling blocks, exceptions not processed at one level are ("vertically") propagated to the containing blocks until they are processed. If an exception is not handled at any level, the program is typically terminated. This multi-level processing model makes processes more resilient and less prone to catastrophic failure.

Exception handling is closely associated with the transactional and recovery properties of processes, see [8,16]. The nested exception handling structure supports recursive recovery strategies and can be used to reason about the transactional properties and correctness of a process, see [11]. In this paper we do not deal with transactional aspect of exception handling.

3.1 Exception Handling in Graph-Oriented Processes

WSFL incorporates exceptions as a particular flavor of transition condition in control links. When an exception is generated at an activity, only those links labeled with the correct exception are activated while all others are disabled. Exception handling in WSFL takes place at the local (activity) level, and its effects are propagated across the process graph by DPE.

Structured exception handling has been proposed for graph oriented process models by Hagen and Alonso [11] in the Opera process support system. In Opera, processes are composed of a series of nested blocks and subprocesses, to which exception handlers are attached. Blocks and subprocess are connected to each other by control links following a model similar to the one described in Section 2. Exceptions propagate vertically through the block and process containment hierarchy, while DPE is enabled within each block and subprocess. The result

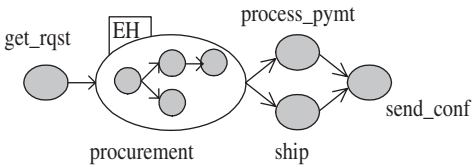

Fig. 3. A hierarchically structured graph: control links do not cross block boundaries.

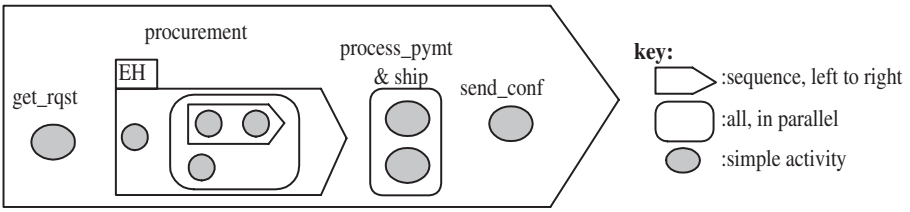

Fig. 4. A fully structured process: structured activities are nested to create more complex execution patterns.

is a layered or structured graph model in which the process structure and the exception handling hierarchy perfectly match each other.

3.2 Exception Handling in Algebraic Processes

Algebraic process modeling languages such as Xlang [17] and BPML [2] are fully structured languages in which the control structure of the process is defined by recursively nesting tasks and primitive control structures. Exception handling is thus natively "structured'. Exception handling blocks are represented by a specialized construct (such as Xlang's "context") to which exception handlers are attached, and which is always properly nested within the overall process hierarchy. The process topology and the exception handling structure are thus perfectly adapted. Figure 4 shows an example of a fully structured process.

Figures 3 and 4 show a similar process using a graph-oriented hierarchical structure and a fully structured approach. Observe that in Figure 3 tasks within a block are not connected to anything outside the block, but blocks themselves are linked to other tasks and blocks. Exception handlers ("EH" in the figures) can be added to any block in both cases (directly or using an additional wrapper).

4 Exception Handling in BPEL4WS

BPEL4WS [6] introduces structured exception handling capabilities for both structured processes and arbitrary graphs. BPEL4WS does not require the process topology to follow the exception handling structure, so it would not be

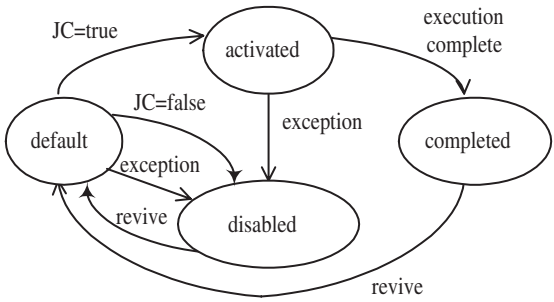

Fig. 5. Activity lifecycle diagram with exception support.

possible to apply the layered approach followed by Opera in integrating exception handling and DPE operational semantics. One of the main design challenges of BPEL4WS was precisely how to perform this integration in a coherent fashion.

This section describes the BPEL4WS exception handling model as implemented by BPWS4J [5]. In Section 5 we explain how the integration of exception handling and DPE semantics is achieved in BPEL4WS. To focus our discussion to the problem at hand, we will begin with a simplified version of the BPEL4WS metamodel, concentrating mainly on simple activities and scopes.

4.1 Exceptions and Activity Lifecycle

We extend the metamodel of Section 2 by introducing activity exceptions and "scopes". Exceptions broadly represent any type of error or unusual processing event occurring at a particular location, that is, at a specific process activity. Exceptions are globally named and can optionally carry exception data. There are three types of exceptions in BPEL4WS:

1. Application exceptions are the ones generated by an application that is invoked by the process. BPEL4WS assumes a normalized representation of application signatures and exceptions based on the WSDL language, see [4].
2. Process defined exceptions. A process may explicitly generate an exception using a *throw* clause.
3. System exceptions are thrown by the process engine in response to a set of possible error conditions, including datatype mismatches, asynchronous termination, etc.

Exceptions cause activities to terminate. Since exceptions represent error conditions, we deem the state of the activity after the exception has been generated to be "unsuccessful" or "disabled", and model this in the modified activity lifecycle diagram of Figure 5 by an additional arc linking the active state to the unsuccessfully terminated state. One more arc is added, from the default state to the disabled one, corresponding to the disablement of activities in the default

state in case their scope has received an exception. Observe that this characterization requires that the value of all outgoing control links be set to false after an exception is generated.

4.2 Scopes

Scopes are sets of activities. Scopes are arbitrary sets except for the limitation that partial overlap of scopes is not allowed, that is, given any two scopes in a process model, they are either disjoint or one is contained (nested) inside the other. A scope can have exception handlers attached to it, with the usual semantics: an exception handler is executed if an exception of the declared type is generated within the scope. Vertical propagation along the scope containment hierarchy occurs, as usual, when no appropriate exception handler can be found in the current scope. The process model is implicitly regarded as the top level scope.

In the model followed by BPEL4WS and implemented in BPWS4J, an exception generated at an activity causes the termination of all activities within the immediately containing scope. It is not possible to continue normal processing within a scope after an exception has been thrown (as allowed in [11], for example). Formally, all inactive and executing activities within the scope are immediately terminated just as if an exception had been locally generated (in fact a special process exception is generated to allow activities and scopes to perform necessary clean up and recovery before being terminated). Naturally, the meaning of "immediate" termination depends on the interruptible nature of each activity. Finally, the state of all activities, except for those already in the completed state, is set to "disabled", and, consequently, all outbound control links are set to false. Termination caused by the generation of a fault within the enclosing context has been reflected in the updated activity lifecycle diagram of Figure 5.

The vertical propagation of exceptions and its effect on control links is illustrated in Figure 6: two similar processes are shown, one with an exception handler on an inner scope, and the other with the handler on the outer scope. Assume that all transition conditions evaluate to true, and that an exception is thrown by the activity shown with the "!" sign. In the left diagram, it is caught by the inner scope's exception handler (EH) and the link originating from that scope is therefore allowed to activate normally; however links originating from inside the scope boundary are disabled (except for those already activated.) In the diagram on the right, the inner scope has no handler and the exception is propagated to the parent scope. In this case, links leaving the inner scope are disabled. The exception is again contained, however, and the final activity at the bottom still runs.

Scopes are treated as composite activities and follow a similar lifecycle. Just like any activity, a scope can be the target or the source of a control link. The activities a scope contains have a control dependency on the scope (an implicit, required control connector); that is, activities inside the scope cannot be activated if the scope itself is not activated. Since we do not require the flow of

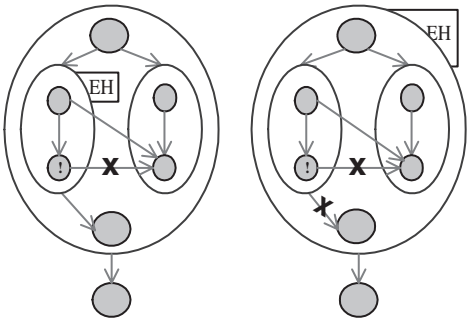

Fig. 6. Scopes and exceptions: an exception is thrown by the activity with "!". Left: exception is handled by the inner scope. Right: exceptions not handled are propagated up to the containing scope. Links are deactivated accordingly.

control to follow the nesting structure of scopes, it is possible that the activities within the scope may be connected by control links to external activities (out of the scope).

A scope terminates when the all activities it contains terminate. An active scope completes successfully when all its contained activities terminate, provided all exceptions generated are successfully handled by exception handlers at or within the scope. If the scope is unable to process an exception (because it lacks the appropriate handler or because a new exception is thrown by the handler itself) then the scope is considered to have terminated unsuccessfully and enters the disabled state.

This construction allows us to maintain the usual activation and termination semantics (and dead path elimination, in particular) for activities as well as scopes, and to freely combine structured and unstructured authoring styles. In the structured style (Figure 7a), scopes are treated as isolated blocks. Join conditions and dead path elimination are used to terminate complete scopes in the same way as activities. In the unstructured style (Figure 7b), tasks within the scope can be linked directly to external tasks; the scope may not be explicitly activated at any time except through activation of the activities it contains. A combination of both behaviors is also possible, as shown in the Figure 7c.

5 Dead Path Elimination as Exception Handling

Introducing exceptions required us to incorporate minimal updates into the basic lifecycle state diagram of Section 2. The updated lifecycle of activities provides two mechanisms with different operational semantics to disable an activity: exceptions and dead path elimination (join conditions). These mechanisms are integrated in the BPEL4WS language by casting false join conditions as a particular type of system exception, the *join exception* which is generated at an activity whenever its join condition evaluates to false. This exception is all that

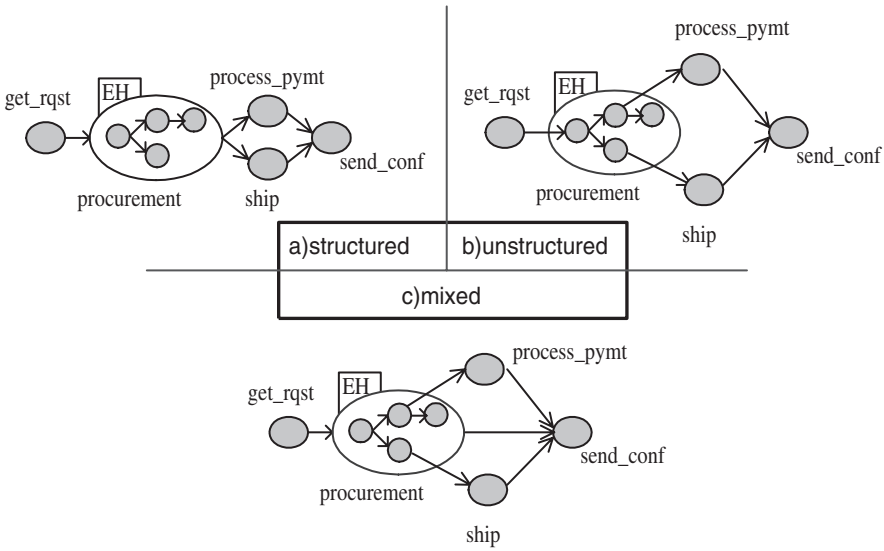

Fig. 7. Different graph oriented BPEL4WS authoring styles. a)Links do not cross scope boundaries. b)Links connect simple activities only. c)Both the scope and its enclosing activities are linked to other activities.

is needed to reduce DPE operational semantics to exception handling. Moreover, join exceptions provide a mechanism to generalize DPE semantics to cover alternative execution and termination models for process graphs. An updated state diagram incorporating this change is shown in Figure 8

To understand how join exceptions can reproduce DPE, consider the arrangement shown on the left of Figure 9. There, an activity is contained in a private scope which is equipped with an empty exception handler for the join exception. The activity, not the scope, is wired to other activities in the flow.

Assume now that the join condition JC evaluates to false and a join exception is thrown. Activity A is set to the disabled state, but the exception is caught and suppressed by the empty handler in the enclosing scope; all outgoing links are set to false. Thus, in this particular setting, the net effect of a false join condition is to disable the activity and propagate the false value of the join condition to all outgoing links. If we now consider a process where all activities are of this type, it is easy to see that we recover standard DPE behavior.

Discussion. Upward ("vertical") propagation of exceptions is a well understood part of the operational semantics of most structured exception handling models. Among other things, the propagation of an exception along the scope hierarchy helps ensure that the effect of an error in one activity reaches other activities in the process as defined by the scope hierarchy. The preceding discussion shows that dead path elimination is essentially equivalent to the propagation of ex-

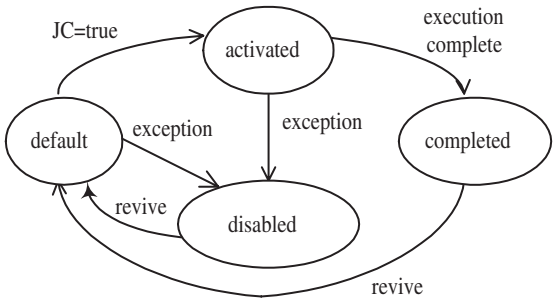

Fig. 8. Activity lifecycle diagram combining DPE and exceptions.

ceptions (join exceptions in this case) along the process graph; that is, DPE is nothing but the "topological" or "horizontal" propagation of the join exception.

5.1 Generalizing DPE

Conditional Tasks. Consider now the arrangement shown in the middle of Figure 9. Again, we have an activity enclosed inside a scope that suppresses the propagation of join exceptions. However, while incoming links target the activity, outgoing links start at the enclosing scope, not the activity itself.

When the join condition evaluates to false, the activity is disabled and the exception suppressed by the empty handler as before. However, since the exception was caught, the scope terminates successfully, enters the completed state and the outgoing links are not disabled. The net effect now is to disable the activity but allow regular execution of downstream activities. This pattern is thus equivalent to an "if" clause in a structured programming language; it also provides a simple rendering of the operational semantics of the switch construct in BPEL4WS in terms of activities, scopes and links.

Observe also that this pattern allows us to separate the synchronization effect of a join from the DPE processing that follows, under the usual graph oriented semantics, when the join condition evaluates to false. Recal that synchronization is achieved because execution does not proceed beyond the join until the values of all incoming links are received. Under the usual graph semantics, however, a false join condition implies not only that the target activity is not executed, but also that all outgoing links are unconditionally disabled. Using a conditional execution pattern, execution can proceed past the synchonization activity regardless of the value of the join condition, depending only on the values of the transition conditions attached to links that originate on the enclosing scope.

Scope-Wide Elimination. In the absence of a surrounding scope that suppresses the join exception, a false value of a join condition triggers the disablement of all activities (except those already completed) within the immediately

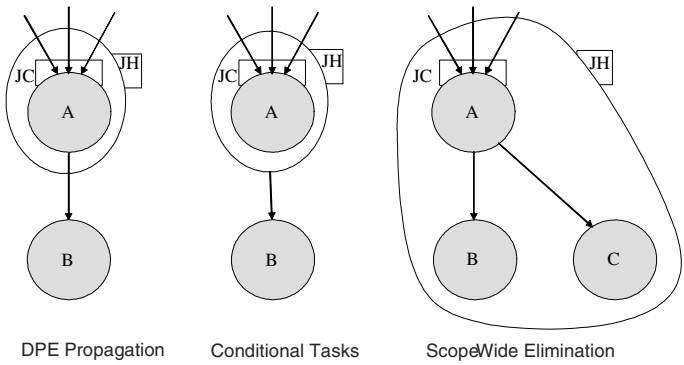

Fig. 9. Three configurations leading to different exception propagation modes.

enclosing context, which might be provided by the process itself. The disabled state of an activity is immediately propagated across the scope regardless of the flow topology. Links emanating from activities inside the scope are disabled (except for those already enabled); links emanating from the scope itself will be enabled if the scope processes the exception. This case is shown on the right of Figure 9, and was also the case exhibited in the example described earlier in Figure 6.

Observe that we can also characterize this effect as a discrete change in the level of granularity at which the process deals with the disabled state, from the individual activity to the wider enclosing scope.

Discussion. The three patterns illustrated in Figure 9 provide three ways of modeling the dependencies between activities in a process model. Under usual DPE semantics, every task in the flow has an implicit dependency on the successful completion of all other upstream tasks. The dependency is enforced by the propagation of the disabled state using DPE.

The conditional execution pattern presented here embodies the opposite assumption, since it prevents the downstream propagation of the disabled state of an activity. In a flow in which each activity follows this pattern, control links and join conditions can be set to false only when the link's transition conditions evaluate to false, that is, only through the effect of the runtime data values accessible to the process instance. The completion status of a task can only affect the execution of downstream activities through its effect on data values.

When the immediately enclosing scope is not restricted to one activity, a dependency between the execution of that activity and all others in the scope is created. This case represents the situation in which one activity is of such critical importance to the successful execution of the scope that it cannot fail without compromising the successful execution of the scope as a whole. Observe also that the dependency stated here is stronger than the one implied by DPE, since it is unconditional and cannot be controlled through the join condition.

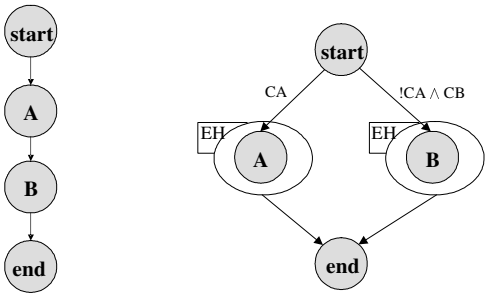

Fig. 10. Graph equivalence of sequence and switch constructs in BPEL4WS.

We note finally that it is perfectly possible to combine the three dependency models outlined above into a single process model. Care must be taken, however, since the local variations in behavior can be very misleading. As an example, observe that the conditional execution pattern effectively stops DPE. The combination of DPE-enabled and conditional activities would lead to a sectioning of the process graph into disjoint areas: inside some of them all tasks would be reachable by DPE, while in others DPE would not apply. For this reason, while BPEL4WS allows arbitrary combinations of scopes and links, it recognizes and provides syntactic support for DPE and non-DPE oriented authoring styles.

5.2 Graph Equivalent of Structured Constructs

The execution patterns described in the previous section can be used to model BPEL4WS structured constructs in terms of equivalent graphs. Once we do this it is possible to deal with the combination of the structured and graph oriented styles in a meaningful way. Consider for example the sequence and switch constructs that we used in Figure 1. Their equivalent graph representation is provided in Figure 10.

Note that for each construct we have added two convenience nodes, start and end, which allow connectors to target and originate on the sequence or switch blocks themselves. A sequence becomes a simple unconditional linear graph, this time without associated DPE behavior. Join exceptions are not dealt with at the activity or sequence level. In the presence of additional (explicit) control connectors targeting an activity in the sequence, a false transition condition would thus cause the entire enclosing scope to terminate. The behavior of a switch block is supported by the conditional execution pattern presented before. The guards in the branches of the switch, CA and CB, define the transition conditions (CA) and ($!CA \ \wedge \ CB$) of the corresponding links. Each branch is enclosed in a scope that prevents the join exceptions generated by false transition conditions from being propagated.

6 Future Work

We are already working on providing mechanisms for getting tighter control on link behavior in faulting scopes. Basically, restricting links originating within a scope to leave it only if the scope completed seems to be a promising direction. Furthermore, BPEL4WS supports extended transactions in tight coupling with exception handling in scopes. Links leaving a scope "too early" result in the need for cascading both, exception handling and transaction rollback similar to what has been described in [12]. We want to investigate the impact on cascadation in situations in which both types of scopes (the one kind described in this paper and the one sketched here) exist within one and the same process.

7 Conclusion

In this paper we have discussed the integration of graph oriented and algebraic (structured) process models in the BPEL4WS language. The key to this unification is an exception handling mechanism that can deal with both algebraic and graph process models. Using this mechanism, we have explained DPE as the handling of a particular system exception under a specific process topology. We have shown how DPE can be generalized by changing the local topology of the process, and how the generalized patterns we obtain can be used to characterize the operational semantics of algebraic control constructs in terms of equivalent graphs.

Acknowledgements. BPEL4WS was a joint effort of the authors of the specification. Satish Thatte and Dieter Roller played a crucial role in designing the integration of the DPE and exception handling mechanisms in particular.

References

1. Process Definition Interchange Process Model. Published on the World Wide Web by the Workflow Management Coaltion as Document Number WfMC TC-1016-P, Version 1.1, at http://www.wfmc.org, 1999.
2. Assaf Arkin. Business process modeling language - bpml1.0 last call working draft. Published on the World Wide Web by BPMI.org at http://www.bpmi.org, 2002.
3. E. Best and C. Fernandez. *Non-Sequential Processes: A Petri Net View.* Springer, Berlin Heidelberg, 1988.
4. Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. Published on the World Wide Web by W3C at http://www.w3.org/TR/wsdl, Mar 2001.
5. F. Curbera, M. Duftler, R. Khalaf, N. Mukhi, W. Nagy, and S. Weerawarana. BPWS4J. Published on the World Wide Web by IBM at http://www.alphaworks.ibm.com/tech/bpws4j, Aug 2002.
6. Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, and Sanjiva Weerawarana. Business Process Execution Language for Web Service (BPEL4WS) 1.0. Published on the World Wide Web by BEA Corp., IBM Corp. and Microsoft Corp. at http://www.ibm.com/developerworks/library/ws-bpel, August 2002.

7. Francisco Curbera, Frank Leymann, Dieter Roller, and Sanjiva Weerawarana. Web Services Flow Language (WSFL) 1.0. Published on the World Wide Web by IBM Corp. at http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, May 2001.
8. J. Eder and W. Liebhart. Workflow recovery. In *Proc. Intl. Conf. on Cooperative Information Systems CoopIS'96, (Bruxelles, Belgium)*, 1996.
9. C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join calculus. In *Proc. 23rd ACM Symposium on Principles of Programming Languages (POPL '96)*, pages 372–385, 1996.
10. Dimitrios Georgakopoulos, Mark F. Hornick, and Amit P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
11. Claus Hagen and Gustavo Alonso. Flexible exception handling in the OPERA process support system. In *International Conference on Distributed Computing Systems*, pages 526–533, 1998.
12. F. Leymann. Supporting business transactions via partial backward recovery in workflow management systems. In *Proc. BTW'95 (Dresden, Germany), Springer 1995*, March 1995.
13. F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33:326–347, 1994.
14. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice Hall., 2000.
15. Robin Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.
16. A. Reuter. Managing distributed applications with contracts. In *Proc. 3rd Intl. Workshop on High Performance Transaction System (Asilomar, CA)*, 1989.
17. Satish Thatte. XLANG. Published on the World Wide Web by Microsoft Corp. at http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, 2001.
18. Wil M. P. van der Aalst, Alistair P. Barros, Arthur H. M. ter Hofstede, and Bartek Kiepuszewski. Advanced workflow patterns. In *LNCS of Conference on Cooperative Information Systems (CoopIS2000)*, volume 19 of *Lecture Notes for Computer Science*, pages 18–29, 2000.

Ratios to Support the Exploration of Business Process Models

Andreas Dietzsch

Schweizerische Mobiliar Versicherungsgesellschaft, Bundesgasse 35,
3001 Bern, Switzerland
Andreas.Dietzsch@mobi.ch

Abstract. In 1999 the Swiss Mobiliar Insurance Company (Mobiliar) started a program to transform to a process oriented organization. To ensure a unified methodical procedure during this transformation a competence center was founded. After the first phase the core business processes were documented. To ensure a continuous usage of this work, a system of ratios was developed to support the process management. This paper introduces the main ratios, how they link to models and how they are used.

Keywords: Business process modeling, design and analysis of business processes, business process verification and validation

1 Background

As Switzerland's first private Insurance Company the The Mobiliar Insurance Company (Mobiliar) provides services for private persons, companies and the public service sector since 1826.

In 1999 a major project was started to implement a process oriented organizational structure. The goal was to achieve a systematic, long-range related development process of the Mobiliar's business potential. This effort was based on a comprehensive look at the requirements of business processes' reorganization and at the reformation of the application landscape. To support this work a methodology for Business Process Reengineering and modeling was provided by a competence center [3].

The methodology's design was driven by the need to assure daily business through every phase of the change process. This is essential because customers are highly sensitive to risky undertakings of "their" insurance company. To reduce risk of changes to the organization "evolutionary changes" and "problem solving based on models" were constituted as principles of work. Thus, every change was planned and based on the actual business solution. The consideration of the principle of model based problem solving particularly enables the definition of activities and the assessment of its consequences without affecting the "real world" [4]. To fulfill requirements resulting from applying these principles, the competence center provides a customized method through situational method engineering [1], [2].

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 291–301, 2003.

After finishing the first transition phase all core business processes were documented in form of process maps and service specifications as defined by the PROMET BPR methodology [9]. Some of the business processes were defined in more detail using the UML. At this state the documentation contains descriptions of more then 50 processes.

Driven by the brisk and vastly changing market situation since fall of 2001 the need for reorganizing the business raises. One idea in this context was to use the modeled processes as instruments for the support of identification of needs for change. Consequently the methodology was enriched by concepts supporting a quantitative analysis of process design. Until today 9 processes were added to the existing documentation. Thereby the enhanced methodology was used for modeling.

The following section introduces the method used for business modeling and the extension for the quantitative analysis of the modeled processes.

2 The Business Process Modeling Methodology

During the project's phases different goals were strived. These are in sequence the phase of process engineering, system requirements specification, and process improvement.

To provide a method, which satisfies these requirements, method fragments were needed that either partly or completely covered these domains. The following section describes, how the specific requirements of each goal were considered by the engineered methodology.

2.1 Method Support for Business Process Engineering

Because of the business process engineering focus in the first project phase, the decision was made to use PROMET BPR. Thus from the organizational design fragment of PROMET BPR the following concepts were selected [3]:

- The process typology, with types of managing, performing and supporting processes
- The concept of process maps as high aggregated process models, which can be recursively refined
- Service specification,
- The concept of the sequence of formulating strategy, goals, critical success factors, and index values

Driven by the need to simplify the communication of the models, the notation as proposed in [9] and [7] was changed. Thus (graphical) formatting options such as colors were used to express semantics such as process types. Using this methodology about 60 processes were defined by process maps and service specifications. Thereby a process map describes the structure of a process, that means its sub processes and the services consumed and provided. An example of a process landscape is shown in figure 1.

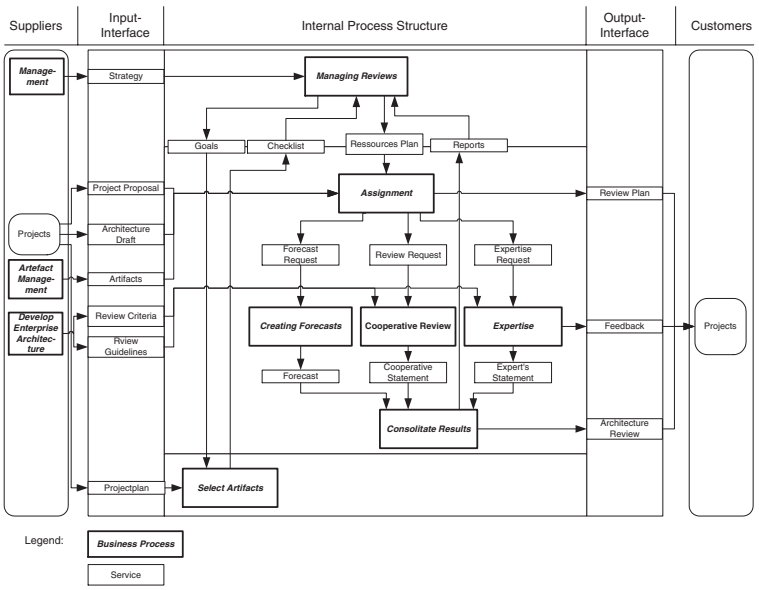

Fig. 1. Example of a Process Map

2.2 Supporting Requirements Specification

The next step was to initiate the adoption of the information systems to the resulting new business needs. For this purpose the system requirements had to be specified based on the newly created business process model. For this project step the UML/RUP [8] was considered as adequate to provide the required method support. The following UML artifacts were selected for modeling requirements to information systems:

- Business Use Case Model containing Business Use Case Diagram and Business Use Case Specification
- Activity Diagrams
- Business Object Model

In contrast to the PROMET BPR fragment, the notations of the UML fragment were not changed. Only a loose coupling of these model elements to those used from the PROMET BPR methodology was defined until fall of 2001. So a process defined in a process map only built a frame for a often complex Business Use Case Model. The associations between the business use cases of such a model contained information about the process structure too. In effect there was more then one place to model such information. Furthermore only basic concepts of Activity Diagrams were used. So diagrams modeled during this time don't contain objects or swim lanes. Corresponding information was placed in the text section of each business use cases' specification.

To overcome these deficits the UML artifacts were linked to the PROMET fragment through some semantic constraints. First there has to be a one-to-one association between business processes of the PROMET fragment and the Business Use Cases of the UML fragment. Thus the traceability between a process map and a Business Use Case was assured.

Furthermore the association types between Business Use Cases were restricted to specialization associations. These are used to express alternatives in realizing a business process. Thus a variant is modeled as specialization of a then so called "abstract" Business Use Case. It defines the inputs and outputs of a process. Its variants are differed by the process realization, e. g. the number of activities or decisions. The Business Use Case Specification document summarizes these characteristics of the process' realization.

Other constraints address the Activity Diagrams. In contrast to the standard UML semantics, they need to contain at least three swim lanes. Two swim lanes are used to place input objects and output objects of a process. Each class of a business object in these swim lanes has a one-to-one association to a consumed or provided service in a process map. At least one swim lane is used to express the responsibility for performing the activities.

After this phase of process modeling the business process model now consists of various diagrams at different levels of detail, represented by Process Maps, Business Use Case models and Activity Diagrams.

2.3 Preparing for Measurement Support

As described earlier, changing market conditions caused a switch of focusing on to process improvement activities. In this context the idea arose to use the existing business process models as starting point for the analysis of change potential. This leads to the demand of ratios for the analysis of the process models and thus requires to define index values and link them to the concepts of the methodology.

In PROMET BPR goals are derived from strategy. A goal describes the services, provided by a business process, and the way, these services will be produced. Critical success factors are referenced to goals by describing potential obstacles that arise while trying to produce the planned services. Critical success factors are implemented through index values, which are formulated based on measurable phenomenon. For the given methodology these phenomena are the elements of a Process Map, Business Use Case model, or Activity Diagram. This relationship is shown in figure 2.

After the described phases the engineered method now consists of concepts mined out of both PROMET BPR and the UML. The integration of these concepts is realized through mainly semantic constraints.

By summarizing information about the realization of the business process through ratios, the process management was able to use the process models as indicators for change potential. The following section describes the system of ratios formulated based on the engineered method.

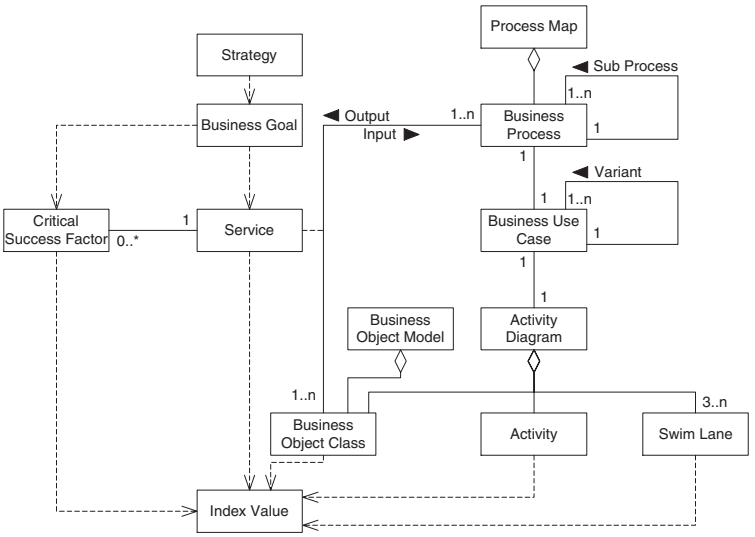

Fig. 2. Linking ratios to elements of business process models

3 The System of Ratios

With the ratios linked to the model elements, the analysis of a business process now can be realized by analyzing the business processes' interfaces, the subprocesses of the business process, and the elements of the Activity Diagram.

To support this analysis heuristics were formulated. Referring to Wallmüller the ratios are described by defining the goal of measurement, the measurement task, the object(s) to measure, and the interpretation of the measured values [10]. Because of missing experiences with such measurements we assumed initial values.

According to the levels of detail as described in section 2.2 the analysis starts at the process model's highest level of detail with the analysis of process maps. To identify change candidates, the first step is to analyze each business processes' interfaces. By relating the number of input objects to the number of output objects the transformation complexity can be assessed (see table 1).

Another characteristic of a business process, which can be determined by analyzing process maps, is the directness of adding value. This ratio is based on the relation between the number of input objects and objects, created within the process but not provided to the customer (see table 2).

If more than one process is identified as a change candidate, the next step is to compare the complexity of the process realization. This starts with the analysis of the Business Use Case(s) linked to the process. In result the standardization of the business process can be assessed (see table 3). We assume that those

Table 1. Assessing the complexity of a transformation

Ratio's Name	***Transformation Complexity***
Goal of Measurement	Determine the complexity of the system of sub processes required to perform the business process' transformation.
Measurement Task	Determine the relation of the number of input objects I and output objects O. Transformation Complexity $= O/I$
Object(s) to Measure	Objects
Interpretation	The relation should be between 0.5 and 2. A value smaller then 0.5 means, that a high number of input objects is transformed into a small number of output objects. We assume that such processes tend to subsume a high number of simple activities aimed to "assemble" a provided service. Further investigation should be on the potential to combine those sub processes to one process. A value greater then 2 means, that a small number of input objects is transformed into a high number of output objects. We assume that such processes hide the complexity of providing services in sub processes and tend to be hardly manageable. Consequently further investigation should be on the potential to split sub processes.

Table 2. Measuring the directness of adding value

Ratio's Name	***Directness of Adding Value***
Goal of Measurement	Determine the steps required to provide a service.
Measurement Task	Determine the relation of the number of input objects I and objects created within the process S. Directness of Adding Value $= S/I$
Object(s) to Measure	Objects
Interpretation	The goal is to design specialized processes. That means to only add specific objects to provide the final service. Consequently we assume that the relation of consumed objects and those created during process realization should be about 3 to 1 (30%).

business processes with a high number of variants should be investigated for change potential.

While the three described ratios can be used to select business processes to change, the second group subsumes ratios to assess the consequences of a business processes' design. This analysis is based on the elements of Activity Diagrams, e.g. the activities, forks and joins, decisions, and swim lanes.

Table 3. Assessing a processes standardization

Ratio's Name	***Process Standardization***
Goal of Measurement	Determine the potential spectrum of the set of instances of a business process.
Measurement Task	Count the specializations of a business use case.
Object(s) to Measure	Specialization associations
Interpretation	A high number of variants indicates a high variance in process instances and thus a low standardization. To ensure efficient manageable processes the number of variants of business process should not exceed three.

The first step is to determine the complexity of process realization. Here the starting point is the number of activities that build a process (see table 4). Deeper analysis should be performed with processes, that are characterized by a high complexity.

Table 4. Complexity of process realization

Ratio's Name	***Complexity of Process Realization***
Goal of Measurement	Determine the complexity of a single business process, sub process or process variant.
Measurement Task	Count the activities grouped in a business process or business process variant.
Object(s) to Measure	Activities within a business process
Interpretation	With a high number of activities a business process tends to be more complex. It is assumed that a complex business process is hard to manage. To keep a business process manageable the number of activities should not exceed fifteen.

If a process realization is characterized by a high complexity it can further be analyzed if it is possible to be reduced. One ratio to indicate such potential is the degree of concurrence of the processes' activities (see Table 5).

Another driver of process complexity are the requirements resulting from coordinating activities within a business process (see table 6). We assume, that the need for coordination is mainly driven by activities performed under different responsibilities. By reducing those activities, the process complexity can be reduced.

Table 5. The degree of concurrence of a processes' activities

Ratio's Name	***Degree of Concurrence***
Goal of Measurement	Determine the cohesion of activities of a business process.
Measurement Task	Count the number of activities embedded in forks and joins P. Determine the relation to the number of activities not in forks and joins A. Degree of Concurrence = A/P
Object(s) to Measure	Concurrent activities in activity diagrams
Interpretation	A low value of cohesion can indicate the merge of two business processes. This can result in obstacles for a focused process management. Potentially the complexity of the analyzed business process can be reduced by dividing the process in two processes.

Table 6. Measuring the demand for coordination

Ratio's Name	***Demand for Coordination***
Goal of Measurement	Determine the amount of coordination effort, necessary to manage the process instances.
Measurement Task	Count the number of transitions to activities in other swim lanes S. Determine the relation to the total number of transitions T. Demand for Coordination = S/T
Object(s) to Measure	responsibility cross cuts in activity diagrams
Interpretation	Crossing swim lanes mainly means cooperation and thus induces the need for communication and coordination. Responsibility cross cuts are potential losses of process performance and should be reduced. We assume that the value of this ratio should not exceed 20%.

As described earlier, we assume that one major driver of process complexity, is the variance of process instances. In Activity Diagrams the ratio addressing this driver is the frequency of decisions to make (see table 7).

In contrast to Franken et al. [5] the introduced ratios are focused on business process model measures. Thus not the characteristics of process instances, e.g. response time, processing time, or throughput, are measured but the impact of certain model structures on the modeled business processes can be derived.

Table 7. Measuring the decision frequency

Ratio's Name	***Decision Frequency***
Goal of Measurement	Indicate the variance possible by realization of the process.
Measurement Task	Determine the relation between the number of decisions D to make within the process and the total number of activities A. Decision Frequency $= D/A$
Object(s) to Measure	Decisions and activities
Interpretation	The higher the number of decisions to make within a process the higher is the variance of process instances. Additionally every decision potentially increases the required process time and the skills required for staffing. Increasing the breadth of possible process instances hinders process standardization. Thus generally the decision frequency should be reduced.

4 Experiences

With the described ratios it is possible to explore the Mobiliar's process models at every level of detail. Thereby most ratios are defined at the highest detail level - the Activity Diagrams (see fig. 3).

Level of Processes Model	**Content**	**Ratios**
Process Map	Services Processes Customers and Suppliers	Transformation Complexity Directness of Value Adding
Business Use Case Model	Variants of a Process	Process Standardization
Activity Diagram	Activities Input Objects Output Objects Forks and Joins Swim Lanes Decisions	Complexity of Process Realization Degree of Concurrence Demand for Coordination Decision Frequency

Fig. 3. Process model elements and the defined ratios

Because the Activity Diagrams are stored in a case tool repository we decided to first analyze this model level. Until now we have implemented scripts to measure the "Complexity of Process Realization", "Decision Frequency" and "Transformation Complexity". We performed an overall analysis over all mod-

eled processes, including those modeled before enhancing the methodology. Additionally we separately analyzed the process models regarding the used methodology version. Knowing that the investigated model basis don't allow a statistical "evidence" we measured the following values over all process models.

The average number of activities per Activity Diagram is 9. Thereby the deviation is 5.8 so that 3 to 15 activities per process can be considered as "normal". The average "Decision Frequency" is 32% which means approx. 3 decisions per diagram. The deviation here is 3. So a "normal" Activity Diagram contains 0 to 6 decisions. For the 9 processes modeled using the enhanced methodology we measured the following values:

Ratio	**Average**	**Deviation**
Complexity of Process Realization	8	3.7
Decision Frequency	40%	38%
Transformation Complexity	1.9	1.2

The high decision frequency is interesting because we conclude that these processes - which are all claims handling processes - are less standardized than the others. To prove that our interpretation is correct we will now work with the process management of these processes. Beside this analysis during the last months these ratios were used by Business Process Engineers to get a quick insight into modeled processes and to guide the process design.

5 Conclusion

This paper has shown how, based on specific process modeling methodology, a system of ratios can support the active use of business process models. Furthermore it can give guidance through the process design. First experiences show that one of the success factors of such an extension of methodology is to focus on few indicators instead of trying to completely represent the modeled processes.

An advantage of the described system of ratios is that there is no need for specific tool support to perform business process analysis as for instance in [6]. Thus the process management is now able to assess the business impact of design decisions originating in the process engineering phase. Supported by the provided methodology, the potential for process improvement can now be analyzed systematically. With this extension the character of business process models changes from a instrument of documentation and communication to an analysis tool, and thus the value of models increases.

Further work will be on the complete implementation of the introduced ratios as well as on the evaluation and improvement of the assumed values. Beyond this a extension of the system of ratios to static structure models is strived. Thus the value of the Mobiliar's models and in particular the benefit of applying modeling to changes processes could further increased.

References

1. Brinkkemper, S.; Saeki, M.; Harmsen, F.: Assembly Techniques for Method Engineering, In: Pernici, B.; Thanos, C. (Eds.): CaiSE'98, LNCS 1413, Springer, Berlin et al., 1998, pp. 381–400
2. Brinkkemper, S.; Saeki, M.; Harmsen, F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering, In : Information Systems 24 (1999) 3, pp. 209–228
3. Dietzsch, A., Adapting the UML to Business Modeling's Needs - Experiences in Situational Method Engineering, Jézéquel, J.-M., Hussmann, H., Cook, S. (Eds.), UML 2002 - The Unified Modeling Language: Model Engineering, Concepts, and Tools, Springer, Berlin et al., 2002
4. Esswein, W.: Modellierung der Verteilbarkeit von Daten, In: Fortschrittberichte VDI Reihe 10: Informatik/ Kommunikationstechnik, Nr. 251. Düsseldorf: VDI-Verlag, 1993, pp. 170–182
5. Franken, H. M.; Jonkers, H.; de Weger, M. K.: Structural and Quantitative Perspectives on Business Process Modelling and Analysis, In: Kaylan, Ali Riza & Lehman, Axel (eds.), Proceedings 11th European Simulation Multiconference ESM'97, The Society for Computer Simulation, San Diego, 1997, pp. 595–599
6. Nissen, H. W.; Zemaneck, G. V.: Knowledge Representation Concepts Supporting Business Process Analysis, In: Proc. of Reasoning About Structured Objects: Knowledge Representation meets Databases (KRDB-95), Workshop of the 19th Annual German Conference on Artificial Intelligence (KI-95), 1995, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-2/nissen.ps, download: 13-08-2002
7. o. V.: PROMET ® Methodenhandbuch für den Entwurf von Geschäftsprozessen, IMG, St. Gallen, 1997
8. Kruchten, P.: The Rational Unified Process: An Introduction, 2nd ed., Addison Wesley, Reading, 2000
9. Österle, H.: Business Engineering Prozeß- und Systementwicklung, Band 1: Entwurfstechniken, 2nd Edition, Springer, Berlin et al., 1995
10. Wallmüller, E.: Software-Qualitätssicherung in der Praxis, Hanser, München, Wien, 1990

Integrating Business Process Reengineering with Information Systems Development: Issues & Implications

Vishanth Weerakkody and Wendy Currie

Centre for Strategic Information Systems
Department of Information Systems & Computing
Brunel University
Uxbridge
Middlesex UB8 3PH

{Vishanth.Weerakkody,Wendy.Currie}@brunel.ac.uk

Abstract. Many organisations in the West have undertaken business process reengineering initiatives with the aim of improving organisational performance. These initiatives inevitably involve redesign and alterations to the existing information systems that support the business processes. The implications for information systems and how an organisation's existing systems can evolve to support a reengineering project is an area, which has been relatively under researched. Although it is recognised that information systems design and development is difficult in BPR environments, there is little overlap between research in BPR and systems analysis, so that there is no shared vocabulary and perspective. This paper examines how information systems redesign can be integrated with business process reengineering through a review of relevant literature and empirical research.

Keywords: business process reengineering (BPR), information systems & technology (IS/IT), legacy systems

1 Introduction

The highly competitive market environment that emerged with the 1990's placed many organisations under increasing pressure to improve performance and reduce the cost of running their businesses. A pattern emerged where companies were having to continuously re-skill their people, reshape product portfolios, redirect resources and redesign processes and IS/IT systems [28][52]. In this context, Business Process Reengineering (BPR) became a popular mode of organisational change and improvement in the 1990's [23][32][19]. However, since the turn of the millennium the West has become somewhat shy of the BPR concept. Many business and IS/IT specialists dislike hearing the word 'BPR' although their organisations are still actively involved in some form of reengineering. Present day examples that result in business and IS/IT reengineering include, Enterprise Integration and Business Process

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 302–320, 2003.
Springer-Verlag Berlin Heidelberg 2003

and Application Outsourcing. Researchers and practitioners today use different interpretations for BPR to suit their environments and needs. BPR is discussed under the various rubrics of business process improvement, business process redesign, business process innovation or business process transformation. Harrington [32] uses the term 'business process improvement' in preference to business process reengineering.

Information systems, supported by the plethora of information and communication technologies, sustain the core business processes in most of today's organisations. The main benefits of information technology has moved beyond the efficiency and effectiveness gains of the 1960's and 1970's and towards strategic advantage which will transform the organisation of the future. Therefore, if real benefits are to be realised from business process change it will often involve redesigning the information systems and information technologies (IS/IT) that support the processes [8].

This paper examines the IS/IT - BPR relationship through a review of relevant literature and exploring practical examples where organisations had to integrated IS/IT change with process redesign. The paper is divided into four parts. First, the following section looks in detail at the IS/IT – BPR relationship, the role of IS/IT professionals in process improvement, and the congruence between BPR and IS development lifecycles. Second, the paper briefly outlines the approach used to conduct the research described in the paper. Third, the issues impacting BPR and IS/IT are explored in the context of two cases presented in section four. Fourth, a synthesis of the factors impeding the integration of business process and IS/IT reengineering is compiled in section five. Finally, the paper discusses the conclusions in section six and suggests areas for further consideration.

2 The Impact of IS/IT on Process Improvement

We are living in a period in which organisations particularly in industrialised nations are experiencing a huge growth in the use of IS/IT [6][25]. Information systems have no independent existence of their own unless taken in the context of an organisation and its business processes [37]. While IS/IT is seen as a driver of organisational change [49], it can also play a central role in the BPR process [35][45]and conversely BPR is already changing the way we view IS/IT. Morton [48] suggests that the largest short-term payoffs from BPR may come from reengineering processes with IS/IT support. Although the relationship between BPR and IS/IT remains a difficult one [56][13], companies are likely to engage in IS/IT enabled BPR [9][26]. The move from mainframe based legacy systems to PC based distributed network systems is considered as one of the most useful aspects of IS/IT in BPR [34], though this is often a difficult task which includes changing application software as well as platforms [38].

Behrsin *et al*., [4] and Ciborra [14] saw the changes faced by the 1990's businesses as being fuelled by IS/IT. More recently, the influence of the Internet has further enhanced the capabilities of IS/IT in the context of business process improvement [60][23][25]. While enabling organisations to implement innovative business processes and helping to improve the quality of operations in terms of accuracy and time scale, IS/IT has provided new methods of working that extend the scope of the organisation [51]. On the other hand, like BPR, IS/IT is having a negative impact on people and it is commonly cited as the main factor in causing the decline in the numbers and role of middle managers [21].

Surveys on the use of IS/IT in BPR programmes confirm that IS/IT is enabling the majority of the BPR initiatives [16][57][58][20]. An increasing number of business managers are looking at BPR as a way of applying IS/IT to their business in order to gain competitive advantage and provide quality products and services to their customers [33]. Similarly, surveys conducted by Willcocks [67] highlights IS/IT and its management as one of the top ten critical success factors for BPR programmes; and in practice over 75% of the top 30% most successful BPR programmes had seen IS/IT as critical for both enabling radical process redesign and supporting redesigned processes, thus, signifying the role of IS/IT in BPR.

Given that core business processes in many organisations are usually supported by IS/IT, BPR will inevitably involve redesign and alterations to these information systems [56][10][1]. These processes and people performing them in many organisations often tend to revolve around legacy IS/IT systems. These legacy systems are sometimes scattered across different hardware and software platforms and can be at different levels of compatibility with each other. Thus, it is recognised that IS design and development is difficult in the context of BPR [13][4][56]. The implications for information systems and how the existing systems can evolve to support a reengineering project is an area, which has had relatively little research. An integrated approach is needed to incorporate BPR changes into the business systems analysis and design (BSAD) life cycle, and to understand how information is used in processes [39]. Unfortunately, there is little overlap between research in BPR and systems analysis, so that there is no shared vocabulary and perspective. However, case studies have revealed that IS/IT redesign will be successful when the political and human issues surrounding IS development are well understood and explicitly managed [66][65].

In today's business computing environment there are a number of different methodologies, tools and techniques for business systems analysis and design and for business process reengineering. But, it is commonly accepted that they do not support simultaneous BP and IS reengineering [56][13][42]. The use of different methodologies for process redesign and IS reengineering results in a gap between business process models and IS/IT systems [13][65]. This may contribute to the failure of BPR initiatives especially in cases where the majority of business processes in the organisation are driven by IS/IT.

2.1 IS Development and Process Redesign: The Relationship

What is the relationship between information systems development and business process reengineering or redesign? The views from published literature vary. Hammer & Champy's [29] view is that IS/IT cannot play an effective leadership role in BPR though many IS/IT people may not agree with this view. A number of authors suggest that IS/IT can support fundamental changes to the underlying processes [56][66][15][36]. Harrington [32] and Davenport & Short [20] promote the idea that process improvement should be combined with process automation. Kaplan and Murdock [39] in supporting this view suggest that it is important to take an integrated look at both process and information flows simultaneously, focusing on how information is used in the process and how people interact with systems on both a formal and informal basis.

IS/IT should be viewed as more than an automating or mechanising force; automating processes for the sake of automation will not lead to significant improvements, instead IS/IT should be used to fundamentally reshape the way business is done [46][20]. Hammer & Champy [29] and Harrington [32] agree when suggesting that automating an inefficient process only helps to produce a 'faster mess'.

Although Ciborra [14] questions the applicability of some of the commonly used IS methodologies to practical business environments, it is stated that existing IS methodologies such as Structured Systems Analysis & Design (SSADM) and Information Engineering (IE) overlap with BPR [47]. Earl's [24] view on the relationship between BPR and IS stresses that systems analysis and BPR have a lot in common and share common methods, and suggests that process thinking is the same as systems thinking. Moreover, some researchers suggest that the initiative to move towards BPR frequently originates in the IS department [44].

A comparison of the systems analysis and design methodologies and the more recent BPR literature reveals that approaches to both these areas are based on the traditional, structured lifecycle approach. The traditional systems analysis and design lifecycle is based on the following stages: preliminary investigations and determination of requirements, systems analysis and design, software development, systems testing and implementation (see [7][18][22]). Similarly, the majority of BPR approaches are centred around a cycle of: process identification, analysis, redesign and implementation (see for example [46][61][45]. Therefore, it can be argued that both BPR and IS development lifecycles share a set of guidelines that are based on the same principles.

Although it is recognised that IS/IT strategy planning and business process change activities should be feeding off each other [56], BPR and IS/IT systems development often proceed independently resulting in a mismatch [64][65]. While some firms have been able to achieve multiple objectives when redesigning processes with IS/IT [20], it can be argued that without IS/IT companies may lose much of its power to transform performance [62]. However, as outlined before reengineering legacy

systems often require more effort in comparison to the rest of the process reengineering work.

The increased level of IS/IT and the complexity of systems used by organisations make the reengineering process even more complex. In such an environment, it is helpful to compare the reengineering process with some form of logical construct in order to understand the IS/IT – process reengineering relationship. In this context, Zachman's [69] use of classical architecture (i.e. a process of constructing a building) to understand and account for the different stages of the IS development process is useful.

Like the different stages of the IS development process, the BPR process involves a number of phases (tasks). With a view of rationalising these phases, Zachman's [69] representation of the IS development process can be extended and placed in perspective with the BP&ISR process (table 1). This places the BP&ISR process in context with a non-technical (constructing a building: as in architecture) as well as a technical (developing a information system: as in IS development) process, and helps to understand and justify the relationship between BPR and IS/IT reengineering.

2.2 The Role of IS/IT Professionals in Process Redesign Environments

While the traditional role of IS/IT people were focused more on technical and systems issues in the past, BPR has provided an opportunity for the IS community to combine the rigour of the IS discipline with organisational change [3]. It is a well-known fact that greater business impact can be achieved if IS efforts are focused on business requirements rather than the simple internal efficiency of business functions [53]. IS people are well positioned to contribute to process redesign because of their expertise, techniques and style of thinking [40], especially by recognising any process design limitations that the BPR tools and techniques may have and establishing ways to improve them [55].

Davenport & Short [20] propose that although few IT groups have the power and influence to spearhead process redesign, they can play several important roles. According to Teng *et al*., [57] IS/IT people can identify the IS/IT related enabling opportunities for BPR. Process flows, particularly manufacturing processes in most organisations are often the result of historical circumstance and should usually be redesigned before further automation is applied, and the IS/IT unit can get involved in process redesign by developing a methodology for IT-enabled redesign [20]. Avgerou *et al*., [1] supports this view and indicate that systems analysis and organisational process design have always been linked. Willcocks [66] agrees, but also reminds us that when IS/IT people are heavily implicated in the change process, the chances of ignoring the more human and organisational, including political and cultural issues in change are high. However, such problems can be avoided by involving the process participants and IS/IT users in the reengineering work where the use of trial and error RAD approach can be more constructive for IS/IT reengineering than traditional approaches such as SSADM [46][65].

Table 1. Classical Architecture vs. IS Development vs. BP&ISR

Architecture	Information Systems Development	Business Process & Information Systems Reengineering
Bubble Charts : Understand the final / overall structure of the proposed building.	Scope / Objective : Understand the business environment that needs automation as well as the aims and objectives of the project.	Project Initiation : Understand the business environment targeted for reengineering as well as the aims and objectives of the overall project.
Architect's Drawings : Architect's representation of the building from the owner's perspective.	Model of the Business : Descriptive model of the relationship between business entities from the users perspective.	Strategic Planning : Examine the goals, objectives & IS/IT strategy of the business, and identify areas for improvement.
Architect's Plans : Detailed representation of the final building from the architect's (designers) perspective.	Model of the IS : Detailed model of the information system from the designer's perspective.	Process Identification : Identification and mapping of processes in the context of the current business environment.
Contractor's Plans : Analyse and take into account any issues constraining the architect's plans.	Technology Model : Analyse & take into account any technology issues constraining the designer's IS model.	Process Analysis : Analyse the identified processes and their supporting IS/IT systems and identify problems and opportunities for improvement.
Shop Plans : Drawing the detailed plans of the components that make up the final structure of the building.	Detailed Description : Writing the computer program (software) that produces the instructions for running the actual systems.	Process Reengineering : carrying out the actual process and IS/IT improvement work (mapping the reengineered processes & sub-processes).
The Building : The completed physical building ready for occupation.	Information System : The actual IS/IT system ready for implementation.	Process Deployment : The reengineered process ready for implementation.

"When developing information systems in the past, companies would first decide how they wanted to do business and then choose a software package that would support their proprietary processes" [20a, p125]. However, with best of breed applications, the sequence is reversed and the business process often has to be modified to fit the system. For example, ERP systems are a good example of generic applications. Their design reflects a series of assumptions about the way companies operate in general. Service providers try to capture best practice scenario from experience and incorporate these into their applications. This often results in the user organisation having to restructure their business processes to fit the requirements of the system [20a]. In this scenario, the IS/IT unit has to play a major role to liase with business managers and users to reengineer and bridge the gap between the proposed enterprise systems and the current business processes.

2.3 A New Definition for Business Process & IS/IT Reengineering

In order to accommodate the increasing role of IS/IT in BPR programmes it may be appropriate to alter the term 'BPR' and hereafter refer to it as 'BP&ISR'. In this context, Hammer & Champy's [29] definition of BPR which propose "the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical contemporary measures of performance, such as cost, quality, service and speed" (p32); can be further refined and a new definition for BP&ISR proposed as follows:

'BP&ISR is the fundamental rethinking and radical redesign of an organisation's business processes and information systems with an aim to achieve significant improvements in quality and service, and optimise costs and productivity'.

The above definition incorporates the role of IS/IT in BPR and covers a wider spectrum of the organisational domain than most definitions proposed in the literature. The following section examines the IS/IT - BPR relationship further as well as other areas that are important for the integrated implementation of BP&ISR.

2.4 The Need for Integrating IS/IT Redesign with Process Improvement

Surveys reveal that IS/IT expenditure in completed BPR projects range between 22-36% of the total cost of the project [27]. However, it is also reported that up to 90% of IT projects fail to meet their goals and less than 25% properly integrate business and technology objectives [50]. Other surveys indicate that UK companies waste 40% of the total amount they spend on IS/IT because the systems they build are not aligned with business strategies [42].

Organisations often tend to use IS/IT to automate accounting, stock control and other routine functions [48]. However, automating of past procedures usually leads to 'electronic concrete', and Martinsons & Revenaugh [43], Remenyi [54], Hutton [36] and Hammer & Stanton [31] all caution that computerising the original organisation thinking behind manual approaches only results in the old procedures to operate faster. Thus, "top management's principal challenge is not to design systems that will process data more efficiently but to create an environment in which people can exploit information more effectively" ([11], p209). "Heavy investments in information technology have delivered disappointing results, largely because companies tend to use technology to mechanise old ways of doing business. They leave the existing processes intact and use computers simply to speed them up" ([30], p104).

Earl [24] expresses the view that IS/IT can be a constraint on BP&ISR, principally because of the nature of legacy architectures where systems have been built in the past to support local, functional needs, which result in limitations for process integration. However, he draws attention to the fact that BP&ISR projects have encouraged firms to analyse the business first, before designing computer systems. Yet, this does not prevent pre-existing IS/IT systems being an obstacle to BP&ISR as UK based surveys by Willcocks [67] highlight technical deficiencies together with

poor IS/IT management as seventh out of ten most significant barriers to reengineering.

While it is clear from the literature that process improvement and IS/IT change should happen in parallel, organisations should therefore try and integrate the two. However, as discussed in section 3.1, many organisations engaged in business improvement projects tend to use different teams, methods and tools for process improvement from IS reengineering. While this scenario is captured in figure 1, the arguments made in this section are further exemplified by the practical cases outlined in the following section.

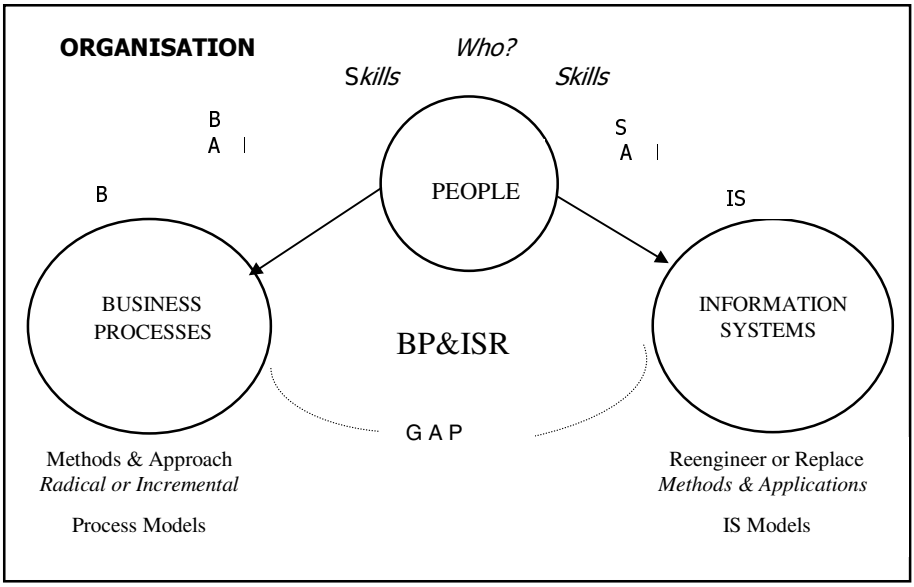

Fig. 1. The Gap between Business Process and Information Systems Reengineering

3 Research Approach

The research described in this paper begins with a review of the relevant literature to examine the role of IS/IT redesign in BPR. To supplement this a substantial case study was undertaken in a large multinational organisation in the UK. This case study was based on participant observation, document collection, formal semi-structured and informal unstructured interviews. The aim of the case study was to investigate how IS/IT redesign is integrated in process improvement. The case study was followed by an action research study in a large South Asian organisation to further explore the IS/IT-BPR relationship.

Four main approaches to data collection were used in both the case study and action research with data coming from multiple sources. These were: formal interviews; informal discussions; observation and document collection.

The philosophy of research adapted draws on the suggestions of Yin [68], Creswell [17], Tesch [59], and Avison *et al*., [2] who encourage the mixing of different data collection strategies.

4 Integrating IS Redesign with BPR: Two Case Studies

This section summarises the case study and action research findings in two organisations that have attempted to reengineer their business processes and IS/IT systems. It outlines the approaches used by these two organisations for redesigning legacy systems and demonstrates how IS/IT redesign was integrated with BPR.

4.1 Organisation 'A' : A Multinational Technology Solutions Provider

This case study looks at the process and IS/IT improvement work undertaken in the UK as part of a very large scale global business reengineering initiative in a leading multinational – technology provider and consulting organisation. The project was promoted and supported by senior management, and consequently the objectives were established at strategic level and seen as fundamental to the long-term organisational success.

The particular aim was to reengineer and standardise the way in which the company manages its customer relationships. Ten key business processes were identified as central to this relationship. A number of systems were being developed to support these key processes, many of which integrate with legacy systems. Of these systems 60% of the reengineered business processes were supported by new information systems, 30% involved integrating new and legacy systems, whilst 10% relied on reengineering legacy systems to match the redesigned processes. By far the largest of these was the customer complaints handling system.

The customer complaints handling process at organisation 'A' was redesigned at a strategic level and handed over to a systems design team to reengineer the supporting IS/IT systems. The approach adopted by the systems development team was to divide the systems redesign work into manageable modules. Direct modifications were made to the system design and program code to match the changes in the business process. Prototyping and Rapid Application Development (RAD) were used in preference to traditional systems development methods such as SSADM or IE, which the IS reengineering team believed consume more time and is more complex to manage although sharing a similar lifecycle as BPR [63].

As the customer complaints handling process had already been reengineered, the task of the IS/IT people was to modify the supporting systems to align it with the

process. Although the systems reengineering team's rapid system development and prototyping approach was practical given the nature and size of the overall reengineering project, the fact that the process had already been reengineered and fixed prior to the IS/IT work was a major handicap to the team. This was the case with the majority of other processes in the project where most of the reengineering work was IS/IT related, while the processes were already reengineered and fixed at a strategic level. This BP&ISR approach had contributed to a large number of IS/IT related problems in the project mirroring the arguments in section 3.

After the IS/IT reengineering work was completed, the team and a few experienced users tested the reengineered system extensively. This was followed by a parallel run with the old customer handling system, after which the reengineered version of the system was delivered to the strategic BPR team for further compatibility testing with the customer complaints handling process.

When analysing the reengineering work at organisation 'A', it was evident that the business process and IS/IT reengineering work had been undertaken in isolation from each other. While the process modelling was done by a high level team of consultants and business planning experts who enjoyed a high level of prestige and recognition, the IS/IT reengineering work was done by systems analyst and programmers who were treated as normal employees. The customer complaints handling system redesign work was a prime example of this scenario where the IS/IT people had less influence on the overall reengineering work, and were expected to simply follow the instructions given by the process modelling team. Unfortunately, this scenario is the opposite of what is considered as best practice in the literature (see section 2).

Perhaps the most significant challenge that organisation A was facing was the difficulty of communicating between the various process and IS/IT teams involved. This was aggravated by the difficulty of mapping business processes to legacy information systems, and the incompatibility between the tools, techniques and methodologies used for BSAD and BPR. However, one of the most valuable lessons learnt from the project was that many of these problems could have been avoided had the process and IS reengineering work been undertaken in integration with each other [65].

4.2 Organisation 'B' – An Organic Based Materials Provider

The Second case study describes the reengineering of key processes in a typical public limited company in South Asia. The case considers the impact of process improvement on reengineering legacy IS/IT systems and the computerisation of non-IS/IT related processes.

In this case, two processes were selected for reengineering by a team of five (including the author) in an action research setting. The first process involved the reengineering of cash and cheque payments to suppliers, and the second process involved the reengineering and introduction of new IS/IT systems to automate the

tracking of engineering projects by engineers and the finance department at organisation B.

As with the previous case study, improvements to the first process required reengineering work to legacy systems that supported the process. Major inefficiencies and weaknesses were found in this legacy system at the process analysis stage. This had to be overcome if any worthwhile improvements were to be made to the process under review. In this context, the reengineering team proposed a solution, which involved a combination of changes to manual activities in the process and the legacy system that was supporting the process. It was clear that substantial savings in the context of staff time, resources and costs were possible if these changes were carried out. However, the reengineering team was unable to carry out these changes on their own and had to involve organisation B's IT function to perform the changes to the legacy system.

Due to the integrated nature of the legacy system concerned, the IT function needed a week to study the proposed changes to the target system. Much to the disappointment of the reengineering team, the IT function was of the opinion that the proposed changes to the legacy systems could not be carried out. The reason given was the risk of, what they described as, a 'chain reaction' to other modules in the integrated system. Nevertheless, the IT staff agreed that the changes proposed by the reengineering team were genuinely effective and admitted that the legacy system concerned, which was introduced to company 'B' in the early 1990's, was highly inefficient and ill-equipped to meet their current business needs.

Analysis of the second process (tracking of engineering projects) revealed that the introduction of a new IS/IT system would significantly improve both the efficiency and effectiveness of the process. Given this, the reengineering team proposed improvements to a number of manual activities and developed a new information system to support the overall process. By introducing IS/IT to the core of the process, it was proposed to speed up and reduce the cycle time of the overall process. While the computerised system involved automatic calculations, transferring of figures, validations, automatically picking up information from the database, the generation of a variety of management information reports and standard printouts; it had a number of data entry standards and controls incorporated helping to eliminate manual errors.

Since the main reengineering effort was focused on computerising the manual activities involved in the process, the work at this stage involved mostly systems design and development. A combination of SSADM and IDEF0 (process mapping diagrams) techniques were used to draw up rough sketches of the 'To Be' process and system models. Whilst the initial analysis and design was done using a structured approach, the latter part of the system development was done using a RAD approach.

When evaluating the process improvement efforts relating to both processes, it is clear that they were centred more on IS/IT reengineering than process redesign. Although the outcome of the first reengineering effort was that the process could not be implemented due to a legacy system constraint, it answers the question set out in

this paper in the context of the position and the significance of IS/IT redesign in process reengineering environments. Likewise, the main reengineering effort in the second process was focused on computerising the manual activities in that process and therefore involved mostly systems design and development. Therefore, it is fair to suggest that both projects (at company 'A' and 'B') helped to exploit the integration of IS/IT with BPR to a large degree.

5 The Main Barriers Impeding the Integration of BP&ISR: A Synthesis of Issues

Although some argue that IS/IT is not necessarily an element of BPR type projects of organisational change, the reality of modern business infrastructure is such that IT occupies a central role [62][25][20a]. Therefore, it is argued that IS/IT has to play a major role for process reengineering to be successfully implemented [56][27][65a].

Issues identified in the literature (in section 3) that impede the efforts for integrated BP&ISR implementation include:

- non-alignment of business and IS/IT strategy;
- poor levels of management commitment;
- constraints imposed by legacy systems;
- risks associated with business and IS/IT change;
- limited scope for team work between business and IS/IT people;
- negative employee attitude;
- red tape and bureaucracy within functionally oriented organisations;
- and lack of frameworks for integrating BP&ISR

The two projects outlined in section 4 further substantiate the above and highlight the close relationship between *BPR and Information Systems*. Among the most significant issues identified in both organisation 'A' and 'B' were the problems associated with reengineering legacy systems. This showed that while most of today's organisations cannot escape the need to change their pre-existing IS/IT systems in the event of a process improvement or reengineering project, IS design and development in the context of BP&ISR can be difficult. The related IS reengineering problems identified, were: the lack of integration between business process and IS/IT reengineering; lack of co-ordination and team work between process reengineering and IS/IT professionals; and the risk associated with reengineering established legacy systems. The influence of the above factors on BP&ISR is captured in figure 2.

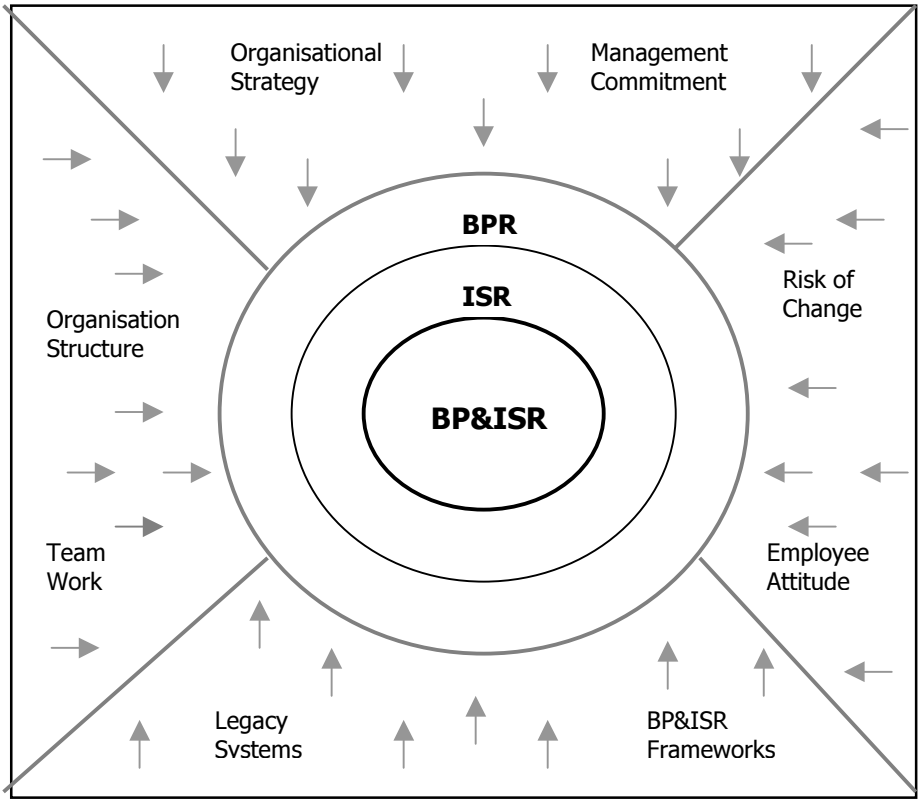

Fig. 2. Factors Impeding BP&ISR

6 Conclusion

Whilst the relationship between business process reengineering and IS/IT remains difficult to understand, it is clear that the reality is BPR and IS/IT often go hand in hand. The literature has demonstrated that both the technical aspects and analysis methodologies for IS/IT and BPR are interwoven. Most organisations engaged in analysing their processes will inevitably have to consider the supporting IS/IT systems. By the same token, an organisation developing a new IS/IT system will have to bear in mind the ramifications for their business processes.

Although the significance of IS/IT in BPR was evident in the literature, most articles do not address this area in adequate depth or provide guidelines for the practitioner on how to carry out IS redesign in BPR projects. Thus, the authors have redefined the definition of BPR and called it BP&ISR to accommodate the reengineering of IS/IT systems that support an organisation's business processes.

Process improvement often involves the reengineering of legacy IS/IT systems as such systems support the business processes targeted for improvement. This paper has sought to explore this relationship by reviewing the relevant literature and comparing this with the realities experienced in two different organisations.

In identifying the positive aspects of this relationship it should be recognised that this is somewhat a generalisation. The case studies help to expand on this, as the two companies discovered different levels of compatibility between redesigned business processes and existing IS/IT systems. In both companies A and B, issues arose as a result of the incompatibility between the reengineered business processes and the proposed changes to legacy systems. It was clear in company A that changes were required in both the process modelling approach and IS/IT redesign approach to facilitate the integration and mapping of process models with new, reengineered and existing legacy IS/IT systems. The isolation of business process and information systems reengineering from each other was clearly the most significant contributor to a number of problems encountered during the process improvement work in this organisation.

Recognising the importance of IS/IT in reengineering, the author has extended Hammer & Champy's [29] definition of BPR to incorporate the role of IS/IT in reengineering, 'BP&ISR', defined as follows:

> 'BP&ISR is the fundamental rethinking and radical redesign of an organisation's business processes and information systems with an aim to achieve significant improvements in quality and service, and optimise costs and productivity'.

The above definition of BP&ISR, presented from the literature, suggests only radical redesign of an organisation's information systems. However, the case studies described in this paper revealed that radically redesigning pre-exiting information systems was not always feasible and was an intricate task. Therefore, in order to accommodate the legacy systems constraints and other key issues impeding BP&ISR in practice, the conceptual definition of BP&ISR above is modified and a new definition proposed as follows:

'BP&ISR is the fundamental rethinking and radical redesign of an organisation's business processes and the *redesign of legacy information systems or implementation of new information systems* with an aim to achieve significant improvements in quality and service, and optimise costs and productivity'.

The main difference between the two definitions lie in the approach used to redesign the organisation's legacy information systems that support the processes selected for reengineering. In cases where the redesign of legacy systems is not feasible, the second definition proposes the introduction of new information systems to support the reengineered processes.

The primary objective of business process reengineering is to achieve improvements in efficiency, effectiveness, productivity and speed of service as well

as cost savings [29]. Arguably the IS/IT reengineering work in process improvement environments should also be inspired by the same objectives. By seeking to achieve these objectives BP&ISR team effectively initiate the process of analysing the organisation's IS/IT systems and the overall IS/IT strategy in general. This process inevitably involves the IS/IT function and thus requires the services of systems analysts, programmers and other technical people. As the case studies show, the overall impact of IS/IT on process improvement is significant and the redesign of IS/IT systems plays an important part in helping to achieve the overall objectives of any process reengineering initiative. By having to *combine IS/IT in process reengineering*, organisations are given the opportunity to rethink their IS/IT strategy and redesign their legacy systems in line with current business needs. Given this, it is suggested that companies can use their IS/IT resources to better effect as part of an integrated 'process and IS/IT improvement' project, rather than under the banner of 'business systems analysis and design'. Moreover, this paper has shown that BPR supports the better exploitation of IS/IT and conversely IS/IT enables process improvement. Therefore, at a strategic level it is argued that undertaking integrated BP&ISR is more beneficial to organisations than isolated business and IS/IT improvement projects.

The case studies have also illustrated situations where IS/IT can act as a barrier to business process improvement (as with company B). Furthermore, the case of company B shows how the integration of IS/IT in process improvement environments can instigate new BP&ISR possibilities. The findings in this paper illustrates some of the complexities which exist in the IS/IT-BPR relationship and how BPR and IS/IT redesign can facilitates each other given the appropriate supporting conditions. However, more research is needed to explore these conditions and find ways in which organisations can integrate IS/IT redesign with process change in practical settings. Given this, further research can focus on promoting ways to incorporate process change into stand-alone IS/IT development projects that are often used to automate inefficient business practices.

References

1. Avgerou, C, Cornford, T & Poulymenakou, A (1995), *The Challenge of BPR to the Information Systems Profession*, New Technology Work & Employment, 10(2), pp 132–141.
2. Avison, D, Lau, F, Myers, M & Nielsen, P A (1999), *Action Research*, Communications of The ACM, Vol. 42, No. 1, January 1999, pp 94–97.
3. Balmforth, A (1996), *Three Dimensions of Business Process Reengineering: The Links to Information Systems and Market Changes*, Proceedings of the 1st UKAIS Conference, April 1996, Cranfield, UK.
4. Behrsin, M, Mason, G & Sharpe, T (1994), *Reshaping IT for Business Flexibility: The IT Architecture as a Common Language for Dealing with Change*, McGraw-Hill, UK.
5. Berrington, C L & Oblich, R L (1995), *Translating Business Reengineering Into Bottom-Line Results*, Industrial Engineering, January 1995, Pp 24–27.
6. Bloomfield, B P, Coombs, R, Knights, D & Littler, D (Eds) (1997), *The Problematic Of Information Technology And Organisation*, In Information Technology And Organisations: Strategies, Networks And Integration, Oxford University Press, Pp 1–12

7. Boehm, B W (1988), *A Spiral Model Of Software Development And Enhancement*, Computer, May 1988, Pp 61–72.
8. Broadbent, M. And Weill, P (1999), The Implications Of Information Technology Infrastructure For Business Process Redesign, MIS Quarterly, Vol.23, No.2, Pp.159–182
9. Byrne, B J (1997), *Bpr For Line Managers*, Idpm Journal, March 1997, Pp 24–25.
10. Carr, D K & Johhanson H J (1995), *Best Practices In Reengineering: What Works & What Doesn't In The Reengineering Process*, Mcgraw-Hill, Ny.
11. Champy, J & Nohria, N (1996), *Fast Forward : The Best Ideas On Managing Business Change,* Harvard Business School Press, Boston, Us.
12. Checkland, P (1991), *From Framework Through Experience To Learning : The Essential Nature Of Action Research*, In Information Systems Research : Contemporary Approaches And Emergent Traditions, Edited By Nissen, H E, Klein, H K, & Hirschheim, R, Elsevier Science Publishers B.V., Holland.
13. Childe, S J, Maull, R S & Bennett, J (1994), *Frameworks For Understanding Business Process Reengineering*, International Journal Of Operations & Production Management, 14(12), Pp 22–34.
14. Ciborra, C U (1991), *From Thinking To Tinkering : The Grassroots Of Strategic Information Systems*, Proceedings Of The 12th International Conference On Information Systems, New York, Pp 283–291.
15. Classe, A (1995), *Business Process Reengineering: Practice What You Preach,* Computer Weekly, July 27, 1995.
16. Coombs, R & Hull, R (1995), *Bpr As 'It-Enabled Organisational Change': An Assessment*, New Technology Work And Employment, 10(2), Pp 121–131.
17. Creswell, J (1994), *Research Design : Qualitative And Quantitative Approaches*, Sage, UK.
18. Daniels, A & Yeates, D (1989), Basic Systems Analysis, 3rd Edition, Pitman Publishing, UK.
19. Davenport, T H (1993), *Process Innovation: Reengineering Work Through Information Technology,* Harvard Business School Press.
20. Davenport, T H & Short, E J (1990), *The New Industrial Engineering: Information Technology And Business Process Redesign*, Sloan Management Review, Summer 1990, Pp 11–27.

20a. Davenport, T.H. (1998), *Putting the Enterprise into the Enterprise System*, Harvard Business Review, July-August 1998, pp.121–131.

21. Dopson, S & Stewart, R (1993), *Information Technology, Organisational Restructuring And The Future Of Middle Management*, New Technology Work And Employment, 8(1), Pp 10–20.
22. Downs, E, Clare, P & Coe, I (1992), *Structured Systems Analysis & Design*, Second Edition, Prentice Hall, UK.
23. Dutta & Segev (1999), Transforming Business In The Marketplace, Proceedings Of Hicss, Hawaii International Conference On Information Systems, 1999
24. Earl, M J (1994), *The New & Old Of Business Process Redesign*, Journal Of Strategic Information Systems, 3(1), Pp 5–22.
25. Earl, M & Khan (2001), E-Commerce is Changing the Face of IT, Sloan Management Review 2001 Vol 43:1 p 64–72
26. Fiedler, K D, Grover V & Teng, J T C (1994), *Information Technology-Enabled Change: The Risks And Rewards Of Business Process Redesign And Automation*, Journal Of Information Technology, Vol. 9, Pp 267–275.
27. Grint, K & Willcocks, L (1995), *Business Process Re-Engineering In Theory And Practice: Business Paradise Regained,* New Technology Work And Employment, 10(2), Pp 99–109.

28. Hamel, G & Prahalad, C K (1994), *Competing For The Future*, Harvard Business School Press, USA.
29. Hammer, M & Champy, J (1993), *Reengineering The Corporation : A Manifesto For Business Revolution*, Harper Collins Publishers Inc., Ny.
30. Hammer, M (1990), *Reengineering Work: Don't Automate, Obliterate*, Harvard Business Review, July/August 1990.
31. Hammer, M & Stanton, S A (1995), *Reengineering Revolution Hand Book*, Harper Collins Publishers, UK.
32. Harrington, H J (1991), *Business Process Improvement: The Breakthrough Strategy For Total Quality, Productivity & Competitiveness*, Mcgraw Hill, USA.
33. Hickman, L J (1993), *Technology & Bpr : Identifying Opportunities For Competitive Advantage,* Proceedings Of The Bcs Case Seminar On Bpr, London, June 1993, In Software Assistance For Business Reengineering, John Wiley & Sons Ltd., UK.
34. Hoffman, Z (1995), Business Process Reengineering: A New Strategic Paradigm Shift In Change Management, Bay Zoltan Foundation For Applied Research, Uae.
35. Ho, J K-K (1996), *Mpsb Research Explained*, Journal Of The Operational Research Society, 47(7), Pp 843–852.
36. Hutton, G (1995), *Bpr: Overcoming Impediments To Change In The Public Sector*, New Technology Work And Employment, 10(2), Pp 147–151.
37. Jayaratna, N (1994), *Understanding And Evaluating Methodologies : Nimsad - A Systemic Framework*, Mcgraw-Hill, UK.
38. IBM (1993), *Redevelopment Methodology Manual, Version 2*, Integrated Systems Solution Corporation Issc, Lexington, Kentucky, USA.
39. Kaplan, R B & Murdock, L (1991), *Rethinking The Corporation: Core Process Redesign*, The Mckinsey Quarterly, November 2, 1991.
40. Khalil, O E M (1997), *Implications For The Role Of Information Systems In A Business Process Reengineering Environment*, Information Resources Management Journal, Winter 1997, Pp 36–43.
41. Knights, D, Noble, F & Willmott, H (1997), *'We Should Be Total Slaves To The Business': Aligning Information Technology & Strategy - Issues And Evidence*, In Information Technology And Organisations : Strategies, Networks And Integration, Edited By Bloomfield, B P, Coombs, R, Knights, D & Littler, D Oxford University Press, Pp 13–35
42. Lissoni, C (1992), *Bridging The Gap Between It Strategy And Ad Reality*, Paper Presented At Ibm's Isei Conference, September 1992, Oxford, UK.
43. Martinsons, M G & Revenaugh, D L (1997), *Re-Engineering Is Dead; Long Live Re-Engineering*, International Journal Of Information Management, 17(2), Pp 79–82.
44. Maull, R & Childe, S (1994), *Business Process Reengineering: An Example From The Banking Sector*, International Journal Of Service Industry Management Ijsim, 5(3), Pp 26–34.
45. Maull, R S, Weaver, A M, Childe, S J & Bennett, J (1994a), *The State Of The Art In Business Process Reengineering In UK Manufacturing Companies*, In Operations Strategy & Performance, Edited By Platts, K W, Gregory M J, Neely A D, Cambridge University Press 1994, Pp 43–48.
46. Mcmanus, J (1997), *If You Want To Succeed In Software Development... Try Rapid Application Development (Rad)*, The Computer Bulletin, Bcs, February 1997, 9(10).
47. Mills, M & Mabey, C (1993), *Automating Business Process Reengineering With Business Design Facility,* Proceedings Of The Bcs Case Seminar On Bpr, London, June 1993, In Software Assistance For Business Reengineering, John Wiley & Sons Ltd., UK.
48. Morton, R (1994), *Business Process Reengineering And Information Technology*, Idpm Journal, 4(2), Pp 10–11.

49. Orlikowski, W J (1996), *Improvising Organisational Transformation Over Time: A Situated Change Perspective*, Information Systems Research , Vol. 7, No. 1, March 1996, Pp 63–92.
50. Ornewsletter (1996), *Why Do It Projects Fail*, Operational Research Society, September 1996, No. 309, Pp 12–16.
51. Parker, M M & Benson, R J (1988), *Information Economics : Linking Business Performance To Information Technology*, Prentice Hall, USA.
52. Pascale, R, Millemann, M & Gioja, L (1997), *Changing The Way We Change*, Harvard Business Review, Pp 127–139.
53. Radley, I P (1992), *Planning For Change*, Proceedings Of The 27th Annual Conference Of The British Production & Inventory Control Society Bpics, November 1992, UK, Pp 37–49.
54. Remenyi, D (1995), *Information Systems Mistakes And A New Focus For Information Systems Management,* Proceedings Of The Second European Conference On Information Technology Investment Evaluation, July 1995, UK.
55. Richards, N (1993), *How Can We Reengineer The Business,* Proceedings Of The Bcs Case Seminar On Bpr, London, June 1993, In Software Assistance For Business Reengineering, John Wiley & Sons Ltd., UK.
56. Stickland, F (1996), *Business Process Change: A Systems Thinking Perspective*, World Futures, Vol. 47, Pp 69–77.
57. Teng, J T C, Grover, V & Fiedler, K D (1996), *Developing Strategic Perspectives On Business Process Reengineering : From Process Reconfiguration To Organisational Change*, Omega International Journal Of Management Science, 24(3), Pp 271–294.
58. Teng, J T C, Feidler, K D & Grover, V (1998), *An Exploratory Study Of The Influence Of The Is Function And Organisational Context On Business Process Reengineering Project Initiatives*, Omega International Journal Of Management Science, Vol. 26, No. 6, Pp 679–698.
59. Tesch, R (1990), *Qualitative Research: Analysis Types And Software Tools*, Falmer, Ny.
60. Timmers, P (1998), *Business Models for Electronic Markets EM* – Electronic Markets, The International, vol. 3.
61. Wastell, D G, White, P & Kawalek, P (1994*), A Methodology For Business Process Redesign: Experiences & Issues*, Journal Of Strategic Information Systems, 3(1), Pp 23–40.
62. Watts, J (1995), *An Introduction To Holistic Bpr,* Business Change & Reengineering, Official Journal Of The Institute Of Bpr, John Wiley & Sons, UK, 2(4), Pp 3–6.
63. Weerakkody, V J P, Tagg, C & Bennett, J (1995), *Bridging The Gap Between Business Process Reengineering And Information Systems: Identifying Key Issues*, Paper Presented At The Bit 95 Conference, Manchester, UK, November 8, 1995.
64. Weerakkody, V J P & Hinton C M (1999), *Exploiting Information Systems & Technology Through Business Process Improvement*, Knowledge & Process Management: The Journal Of Corporate Transformation, Vol 6, No. 1, March 1999, John Wiley & Sons, Ltd., UK
65. Weerakkody (2001), *Identifying Potential Barriers To Business Process & Information Systems Reengineering In Sri Lanka*, Phd Thesis, University Of Hertfordshire, UK, January 2001)

65a. Weerakkody, V. and Currie, W.L. (2002), *An Investigation of the Social and Cultural Issues Influencing Information Systems Change in South Asian Countries,* Proceedings of the UKAIS Conference, April 2002.

66. Willcocks, L (1995a), *It-Enabled Business Process Reengineering: Organisational And Human Resource Dimensions,* Journal Of Strategic Information Systems, 4(3), Pp 279–301.

67. Willcocks, L (1995b), *False Promise Or Delivering The Goods? Recent Findings On The Economics And Impact Of Business Process Reengineering,* Proceedings Of The Second European Conference On Information Technology Investment Evaluation, July 1995, UK.
68. Yin, R K (1994), *Case Study Research - Design And Methods*, Second Edition, Sage Publications, London.
69. Zachman, J A (1987), *A Framework for Information Systems Architecture*, IBM Systems Journal, Vol 26, No. 3, pp 276–292.

Undo in Workflow Management Systems

Alessandra Agostini[1], Giorgio De Michelis[2], and Marco Loregian[2]

[1] University of Milano, DICO, via Comelico 39
20135 Milano, Italy
agostini@dico.unimi.it
[2] University of Milano Bicocca, DISCo, via Bicocca degli Arcimboldi 8
20126 Milano, Italy
{gdemich, loregian}@disco.unimib.it

Abstract. Workflow Management Systems are one of the main technology for supporting Business Processes and they need to be as flexible as possible. One relevant issue arising from integration between WfMSs and corporate Information Systems is that of undo strategies, policies and mechanisms. In this paper the state of the art for the undo problem and the solution adopted in the framework of the MILANO Workflow Management System are presented.

1 Introduction

The paper by Schmidt and Bannon [1] which opens the first issue of 'Computer Supported Cooperative Work: an International Journal' argues for the relevance of articulation work within cooperative work arrangements. Articulation work deals both with the meshing of tasks and performers within a cooperative work process and with the interleaving of different processes within the work time of a performer. Moreover, it deals with the continuous changes of cooperative work arrangements. Therefore, systems supporting articulation work must on the one hand liberate workers as much as possible from the routine work they need for coordinating themselves (script) [2]; on the other, help them to be aware of the situation where they are performing and to negotiate new cooperative work arrangements whenever a breakdown occurs (map) [2]. Finally, they need to be open to continuous changes, in order to support a continuous updating of both their maps and their scripts. If we restrict our attention to the main technology supporting Business Processes, i.e. Workflow Management Systems (WfMSs) (see for example [3], [4], and [5]), the above observations translate into a quest for *flexibility*. That is, any WfMS should be oriented to making the organization as flexible as possible and to supporting its changes. On the one hand therefore, WfMSs should allow end-users to temporary change the flow of work in order to handle effectively exceptions and breakdowns. On the other, it should be possible to implement changes of the workflow model into the already running workflow instances. In accord with the above considerations, the MILANO WfMS prototype (see [6], [7] and Section 3) provides as much as possible general solutions to the previous issues.

Moreover, WfMSs need to be integrated with the tools used by the actors of a process: productivity tools; specialized systems (e.g., CAD, graphic packages);

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 321–335, 2003.
Springer-Verlag Berlin Heidelberg 2003

information systems and databases; e-mail and other communication systems. The complexity of exception handling in a WfMS heavily depends on the type of tools used together with the WfMS. In particular, if a user updates an information system or a database, then managing an exception may request to cancel her changes.

Therefore, with the above reflections, we started to analyze in deep the specific needs of users that have to go back from the current state to a previous state of the workflow (undo). It emerged that this is not the simple undo we have to manage in productivity tools, since it does not deal just with undoing the previous action the user did, but rather with undoing some actions performed by different users. Some hints on how to better support these cases can be found in the recent literature on collaborative undo, presenting interesting concepts and solutions (Section 2). Even if these solutions focus mainly on synchronous cooperation, the different proposals fit quite well with the integrated support we envision for facing the undo problem in WfMSs. Section 3 outlines the features of the MILANO WfMS that are relevant for the treatment of undo and the basic concepts and formalisms adopted in our system. Section 4 presents the solution we designed for the undo in WfMSs while in Section 5 a simple example is used to explain our solution. Some open problems and further research directions are discussed in the Conclusion (Section 6).

2 Overview of Undo Models

Various applications provide the possibility to recover from unforeseen errors or reverse inadvertently issued commands and many proposals already exist to solve the undo problem both for single-user (e.g., [8], [9]) and groupware applications (e.g., [10], [11], [12]). In particular, for systems supporting groups of co-workers the problem is more critical than for applications supporting single individuals since users might be especially afraid of destroying artifacts created by colleagues. Typical examples of collaborative systems in which this problem is particularly relevant are group editors (e.g., [10], [11], [12]); in these systems the real-time/synchronous nature characterize more than other aspects the problem and the proposed solutions. However, when working with WfMSs, further issues become relevant (e.g., [6], [7], [13], [14], [15], [16], [17], [18], [19], [20], [21]). For instance, in WfMSs operations can propagate within separate organization sectors and can have different impacts on individuals and the whole organization. In the following main strategies and models for facing the undo problem will be shortly presented.

In groupware a first distinction can be made between *global* and *local* scope for undo, depending on whether a user undoes a particular step executed globally, no matter who issued it, or a step performed by herself, in a so called local context [12]. Of course, cooperative tools must deal with both contexts [22]. Global group undo always leads to a former application state since also actions performed by different users are undone to set back the system to a state it was at in a certain moment. In contrast, local group undo is more complex because, in most cases, it does not lead to a former application state. Only actions performed at a local site, and thus by a specific user, are undone and this often leads, when other users' operations are interleaved with local ones, to a new and previously un-reached application state.

A core concept for modeling undo strategies is that of *timeline* [22] from which different undo modes follow. In [22] the authors give an exhaustive description of this

concept and of the issues arising from it: divergent timelines, causality and side effects, multi-level and interleaving timelines. Every application can be described by its history, meaning the sequence of states it passed through from its start till the current state. Formally a timeline can then be defined as a directed acyclic graph with a single root (application start; see [22] for further details), since otherwise a repeated redo request could produce loops in the system. Undo and redo operations are meta-commands, meaning they just manipulate history and not other objects in the system. On these bases three undo strategies can be defined [10]: single step, chronological and selective. With a *single step* undo strategy [23] a user undoes only the last locally performed command. If more than the last command is kept in history then the single step undo can be repeated until no further previous information remains. With respect to the *chronological undo* strategy, it gives the ability to automatically undo a certain set of commands performed in a particular temporal order, from the latest back to the selected one. Finally, the *selective undo* [8], [9], [23] allows undoing any command in history. It is worth to underline some differences among the above models. A single step undo in a local context might correspond to a global selective undo since even if an action was performed last on a site maybe some other actions were issued by other people sharing the global view. Moreover, both selective and chronological undo directly refer to a particular command in history but, while for having a chronological undo all later commands have also to be undone, a selective undo only affects that particular command and, as a side effect, a new state could also be produced. A chronological undo needs an explicit and precise timing to be defined for the system: in case this is not available, selective undo looks the most suitable strategy.

From these strategies two interesting undo frameworks follow: linear and non-linear undo. In a *linear undo* model [23] actions can be undone only in a single step or in a chronological fashion, meaning that to undo a particular command a user has to undo all the executed commands following it. In a *non-linear* undo model [24] selective undo is made possible so that any arbitrary command in history can be undone without undoing other ones. Implementing a non-linear model, particular attention has to be paid to every command semantic and effects since interferences could be possible and some later commands might actually rely on the one being canceled. These concepts are better illustrated through a couple of graphical examples.

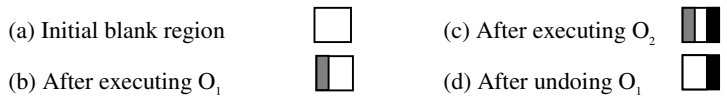

Fig. 1. Example of Non-Linearity

Figure 1 shows the possibility to undo an arbitrary command not affecting the later ones. Initially the region is blank. O_1 draws a grey rectangle in the left side of the region and then O_2 draws a black one in the right side. O_1 is later undone and the grey rectangle is canceled without the need to erase the black one which was chronologically drawn later. Instead, Figure 2 illustrates the effect of undoing a command logically related to a later one. The region is initially blank. O_1 draws a grey rectangle on the left side of the region and O_2 *fills the remaining part* of the region with black color. When O_1 is undone then also O_2 has to be re-evaluated since the area to fill is now different.

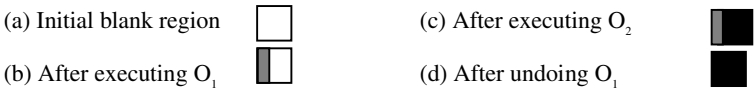

Fig. 2. Example of relation between commands

Considering the database field, the problem of restoring a previous database state has already been faced (see for example [19], [20], [25]). Many proposals exist and most of them can easily be integrated with previous undo models. The most common approach is based on the requirement of being able to undo, redo, or complete *transactions* [25]. A transaction being the unit of locking and recovering which appears to users as an atomic action. ARIES ([26], [27]) is one of these recovery methods. It can be easily extended to persistent object oriented languages, transactional-based operating systems and, as we will later show, also to suitably designed workflow management systems. ARIES has already been implemented in such systems as OS/2, DB2, Starburst, and Quicksilver. The key ideas of ARIES are that recovery repeats history (including actions from transactions that did not commit) and then undoes the actions of transactions that did not commit before the crash (these are called "loser transactions"). It logs the progress of a transaction, and its actions that cause changes to recoverable data objects. The log file ensures either that the transaction's committed actions are reflected in the database despite various types of failures, or that its uncommitted actions are undone. The write ahead logging protocol asserts that the log records representing changes to some data must already be on stable storage before the changed data is allowed to replace the previous version of that data on nonvolatile storage. That is, the system is not allowed to update the nonvolatile storage version of the database until the log records which describe the updates have been written to stable storage, each read and write to a data item by a transaction is blocked until the last transaction that wrote to it has terminated.

3 The Basis of the Project

The work we made on collaborative undo is embedded on the MILANO WfMS prototype ([6], [7]). The MILANO WfMS is a component of a more comprehensive groupware platform whose approach to work practices is based on a situated language-action perspective ([28], [29]) and whose main aim is to support its users in being aware of the history they share with people with whom they cooperate and of the activities they are committed to perform in the future (for more details on the MILANO platform see [30]). The MILANO WfMS, in addition to other usual WfMS services, allows its users to perform not only in accordance with predefined models of work processes, but also when facing unforeseen exceptional situations. Moreover, users are able to modify a procedure definition and, for some classes of changes, directly apply the changes in running procedure instances without fair of incorrect executions or incongruent outcomes. In this paper we focus on those functionalities devoted to handle the cancellation or the revision of some previously executed actions. In the following, we briefly report those aspects of the MILANO WfMS

strictly related to the focus of the paper. As in the majority of WfMSs, the MILANO WfMS allows to define models of the work processes in order to delegate the duty of coordination as much as possible to the WfMS. We choose—for modeling workflows in MILANO—a class of Petri Nets named Elementary Net Systems (ENS) [31]. To simplify the design of workflow models, we allow designers to disregard at the design phase all possible exceptions, which usually occur during the execution of a work process. In fact, MILANO allows its users while executing a work process to create additional paths (jumps) connecting two states of a workflow. In short, whenever the performer of an activity cannot act in accordance with the model, she can jump to another state from which the execution can progress again. We distinguish various categories of jumps on the basis of two main criteria, so that different jumps (exceptional paths) can be associated to different cases and responsibilities. First, jumps are distinguished with respect to the direction of the jump in forward and backward jumps. Forward jumps cause the skipping of some activities while backward jumps involve going back for re-executing or perfecting some activities. Second, jumps are distinguished depending on the degree of concurrency of the skipped or canceled activities. When there is a single running activity we provide strongly linear jumps, while when is necessary to stop several concurrent activities we allow weakly linear jumps. Combining the two above criteria, we therefore allow four classes of jumps: strongly linear and weakly linear backward jumps; strongly linear and weakly linear forward jumps. In this paper we go into backward jumps; as we will better see in next Section, strongly linear backward jumps are the basis for providing users a linear undo mechanism, while weakly linear backward jumps allow a selective undo function. The MILANO WfMS allows associating to each different jump a different responsibility, in accordance with the roles and rules of the organization. For example, strongly linear backward jumps can be under the responsibility of the actor herself. Other policies may be embedded in MILANO. When a breakdown occurs, if necessary MILANO automatically opens a conversation for requesting and discussing with the responsible role the authorization for the jump. More about the characteristics of the MILANO WfMS can be found in [6] and [7].

Let us now recall the basic terms and notations of workflow modeled by using ENS; main concepts will be only informally described. The activities composing a workflow process (formally called transitions) are represented by boxes. The conditions related to the execution of the activities are called places and are represented by circles. There are directed arcs from the input places (pre-conditions) of a transition to the transition, and directed arcs from the transition to its output places (post-conditions). There are two kinds of arcs: input arcs that connect one place to one transition, and output arcs that connect one transition to one place. The state of a place, called its marking, is represented by the presence or absence of a black dot (token) within the place. The presence of a token in a place means that the condition holds. A transition is enabled (i.e., the activity may take place) when all its pre-conditions and none of its post-conditions hold. At the beginning of a work process, i.e. when a workflow model is selected and initiated, at least one transition has to be enabled to grant the starting of the process itself.

To implement our new prototype, we build our system in such a way to be as compatible as possible with already existing Petri Nets based systems. In fact, there are many tools providing useful functionality such as: drawing and editing a net; simulating the net behavior; checking some specific structural characteristic; etc. If

the net describing the work process is described in a standard interchange format usable both for the specific purposes of the new system and for providing the additional services available in other tools. Then, it will be only necessary to develop the specific functions of the new system. We adopt the interchange format defined by Humboldt University in Berlin (see [32], [33]) in which a description of a Petri Net consists of two parts. A general part, which is independent of the specific version of Petri Nets: *Petri Net Markup Language* (PNML). A specific part for defining each net own property: *Petri Net Type Definition* (PNTD). Extracting the typical features of Petri nets into a PNML results in a uniform file structure for all kinds of Petri Nets. For particular net types, it remains to fix the special features of this type in a PNTD.

The developed undo module gives functionalities just to simulate the enactment of a workflow expressed in PNML. What it provides to the user is the possibility to visualize such a process with a graphical representation and to enact (start) as many procedures as possible. It also gives the possibility to simulate users behaviors creating as many clients as possible. Each client gives to procedure's actors a whole set of possible actions: do, undo and redo. A simple but powerful mechanism for document management is also given: forms can be edited and saved in different versions.

4 Undoing Actions in a WfMS

The undo solutions reviewed in Section 2 focus on real-time applications like group-editors. WfMSs are quite different, since they are multi-user asynchronous applications working in a distributed environment (the complex case: there are also WfMSs with simpler architecture). The main difference is that different people work on different tasks at the same time and so various possible undo situations can emerge. We distinguish two main cases: (1) a user needs to undo an action of her own; and (2) a user needs to undo some action(s) that were performed by other users. As recalled in Section 3, the previous MILANO version had a sophisticated system for negotiating authorizations, and in case (2) was always required an authorization. However, in the new MILANO WfMS, a user can directly undo any previously executed procedure step—even if it has been performed by a colleague—without any authorization. In fact, we claim that when skipping some procedure steps (forward jumps) the authorization is absolutely necessary, whereas in all backward jumps we could simplify the authorization mechanism. Therefore the undo operation is simply notified to the responsible person/people (e.g., the performer of the canceled activity). We call this new category of jumps, without the request of authorization, *revised* strongly/weakly linear backward jumps. This way no hierarchy is defined: a clerk can freely undo an activity performed by her boss.

The algorithm for undo we developed has its roots in two earlier works ([22] and [24]) which analyze undo from different point of views and present algorithms that can be usefully merged to be applied to a WfMS. The resulting model is object oriented and non-linear. Objects handled by the systems are transitions mapping workflow activities (and sometimes the whole net itself). The main problem concerning non-linear models is to determine consequential relations among commands, and then deciding which particular commands should be undone after

undoing another one [24]. In order to face this problem, firstly we distinguish between active and inactive transitions. Only an active transition can be undone and only an inactive transition in an undone state can be redone. Secondly, we define the following rules specifying the states of transitions and undo propagations:

- A just completed transition is active and in the done state. All transitions (if they exist) preceding it in the net flow are also active.
- An undo command makes an active transition inactive. All the transitions completed successively and following it in the flow become also inactive; all preceding active transitions will keep their active state.
- A redo command makes an inactive transition active. All transitions (if they exist) preceding and following it in the net will keep their state. Redo commands can only be applied to the uppermost inactive transition in a line.

Some additional considerations are necessary regarding alternative and parallel flows that will be discussed in the following. A work process might contain the possibility for choosing among different alternative (sets of) tasks to be performed from a particular point on. In Figure 3 left two alternative paths (A, X and B, Y) are shown.

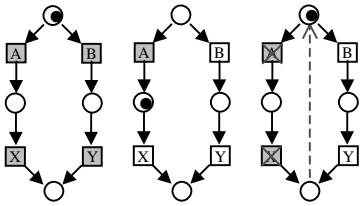

Fig. 3. Example of alternative activities

In Figure 3 middle, a choice has been made during the work process and the transition A has been performed while transition B has not. If after doing so there is an undo request for transition A the choice becomes active again and another evaluation has to be made to decide again which branch has to be followed (Figure 3, right). This undo command is fully supported by a linear undo model (i.e., either chronological or single step) and therefore using MILANO *revised* strongly linear backward jumps.

When two or more activities are not directly related one to another they can be performed at the same time from different people. In a well structured net two or more concurrent or parallel branches in a net start from a transition (S) (see Figure 4 left) having two or more directed arcs outgoing to two or more places. Every place will then be a condition starting a parallel branch. Two or more parallel branches will possibly rejoin in a single transition (E) having them entering in it. Parallel branches can of course be defined even inside other parallel branches.

What happens when an undo request has to deal with concurrent activities? In the simplest case the undo can regard a transition before the flow being divided into parallel sub-flows. That is, for instance, starting from the case shown in Figure 4 left, undoing transition S. In this case, all the already performed transitions in all concurrent branches have to be canceled and a chronological undo mechanism or the

multiple appliance of the single step model completely fit user needs. This is exploited in MILANO via *revised* strongly linear backward jumps. When, on the other hand, we start from a state preceding the re-joining of the parallel branches and the undo regards a transition within a particular parallel sub-flow; then the undo must affect only that particular branch of the net. All the other activities within the parallel branches will not have to be redone. In this case, neither chronological nor single step models are sufficient, due to the concurrent activities that are not involved in the undo. Therefore a selective undo model must be provided; this mechanism is still implemented in MILANO through *revised* strongly linear backward jumps. Finally, a further situation should be considered. An undo request involving a transition within a parallel branch, but starting from a state in which the net has passed its whole parallel section. Suppose it is the case shown in Figure 4 left: transitions S, A, B, and E have

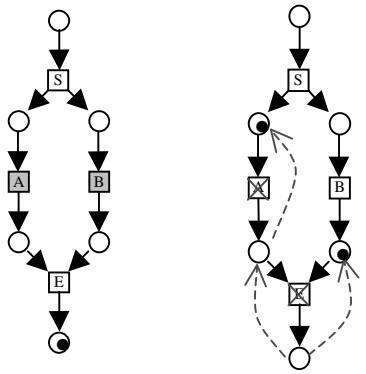

Fig. 4. Example of concurrent activities

already been performed. Transition A has to be undone then also E has to be undone while transitions S and B will not be affected from that request. To set the net in the required state the token now placed immediately after E has to be doubled to fill both places over E and then the one on the left side will have to move before A (see Figure 4 right). Also in this last complex case, we have to support a selective undo mechanism. Moreover, weakly linear backward jumps are not sufficient to support this case (since, in short, they do not allow augmenting the tokens) and therefore a further category of jumps, *weakly relaxed* backward linear jumps, has been added to MILANO to support this situation.

In the new MILANO WfMS prototype a logging mechanism has been developed to better handle the undo commands. It is provided both for document editing and management and for operations such as start, completion, release, undo and redo. This is the way the system is able to trace workflow evolution and also to store information on who performed any task and when. Each activity has its own local history and collects information on documents' versioning (when a document is shared among multiple activities these notes are accessible from each step). In particular, once an activity has been completed, the name of the actor who did it is logged so that, in case the activity has to be redone after an undo command, she will receive the request to redoing it. Whenever an undo is made, a motivation has to be specified and it is also

logged. This way, not only time and author of undo are traced but also the reason for undoing is maintained. A global log for the whole system is also provided tracing the whole evolution from a central viewpoint still recording data from each activity but keeping them for a general overview of the procedure. It could just be seen as the complete set of activity records but it collects also information that are not activity related such as new actors entering the system or new procedures being started, interleaving with the ones already running. Thanks to it, when all activities composing a work process have been performed and the work process is completed, still in MILANO all activities can be reviewed or revised.

Whenever a transition is enabled this means that the task related to it has to be performed by someone. All enabled transitions have to be shown in some kind of to-do list according to the role or the personal reference specified for the transition. In case a role is specified (and no personal reference) then the task related to the transition will be accessible to different people at the same time. The first person that chooses to perform the specified activity, starting it, locks the access to it. However, if a user cannot complete a previously started activity, the user can simply decide to release it and other actors will be able to re-access and continue the activity. It should be underlined that, a re-started transition is not reset to its initial situation. Quite the contrary, if some work was already performed then it is preserved even after the release operation together with every saved document. This service is made possible because each operation performed on activities is fully recorded by the system.

The ARIES algorithm ([26], [27]) briefly described in Section 2 is just one of the well-known approaches that could be inserted within an undo framework to handle database transactions from inside a WfMS. In our prototype we manage to track changes on documents and thus we grant a reliable mechanism for storing and restoring data evolution. We can think of handling interactions between the WfMS and databases in a way similar to the one adopted for documents. Tracing operations arises then to an important role in the system. This should make even clearer the meaning of the previous mentioned concept of timeline [22]. Moreover, activity definition sets boundaries and limits for transactions sets to be committed. All transactions within a single activity can be grouped together and considered as a single complex transaction. ARIES is totally able to robustly handle complex transition and such a packing mechanism could easily be integrated in MILANO. While series of completed activities can be safely logged and undone through sequences of simpler undo operations (relying on database robustness), the problem with ongoing activities being interrupted could concern broken transactions segmentations: actually the mechanism of grouping “smaller” transactions into activity-bounded sets leaves the problem to database management system. A transaction is an atomic unit and can thus not be divided. Until the activity is not completed then the effects on the database are not final.

Another interesting approach is that of Leymann [34], [35] with his analysis of transaction models and the subsequent notion of spheres (atomic spheres and spheres of compensation). A compensation sphere (see [34]) is any collection of activities of a process model such that finally either all activities must have run successfully, or all activities must have been compensated. Each activity within a compensation sphere or the whole compensation sphere itself is associated with an activity called its compensating activity. The basic mode of undoing a compensation sphere is to schedule the compensating activities of all activities within the sphere in an order that is the reverse of the order in which the proper activities of the compensation sphere

have run. In this model, compensations are thought to be customizable, meaning that relations among activities can be defined. It is also a working example of how database management can be embedded in workflow-based applications. Our undo solution is coherent with Leymann's approach, for what regards document management and transactions on databases or information systems, since undo perfectly compensates the executed activities. However, Leymann's work deals with more general activity types involving irreversible actions as, for example, buying a machine. We have still not considered this kind of problems, since we think that are not so directly related with WfMSs. As we will further discuss in the Conclusion, the problem of irreversibility of the activities of a business process is a general problem of process design (involving all its dimensions: organization, finance, human resources and technology) and the Leymann's approach can be extended to become the basis for characterizing *robust* business processes.

5 A Simple Scenario of Use

In this Section we will use a simple work process, the management of the creation of a new web site for of an organization, to better illustrate the main characteristics of our undo module. Figure 5 shows its graphical representation. It starts with the activity *Buy domain* which reserves the name for the coming web site. At this point three concurrent tasks—each one composed by one or more activities—are started. In particular, concurrently to the site design and development phase some activities devoted to allow a reasoned choice between an in-site dedicated web server and a subscription of an external hosting are performed. While a technical person is responsible for defining the HW and SW characteristics of the server and for asking various cost estimations, an administrative person is in charge of collecting the providers' prices for the external hosting option. After that the costs of both alternatives are calculated the manager responsible for the procedure can choose between them. At this point either a PC is ordered, received and paid (some administrative person will be responsible for these aspects) and a technician installs a Web service on it or, more simply, the hosting subscription is made. As soon as the design and development of the site is completed too then the web site is finally published. Some simple cases of undo will now presented referring to the previous sample process.

Jill is chief manager at ACME Co., one of the most important factories of frozen pasta in Italy. At present, ACME Co. has a quite simple web site. Therefore Jill decides that, in order to increase the business, it is necessary to build a totally new web site for improving the on-line selling services. Therefore, Jill initiates the predefined workflow handling the creation of a web site and the activity of requesting and buying a new domain name is allocated. After few moments Jane, an ACME secretary, is informed that a new activity *Buy domain* is in her to-do list (for accuracy sake, as well as in the to-do list of all ACME secretaries). She decides to perform it and therefore selects and starts the activity. She connects to Internet to perform her task and, with few clicks of the mouse, is able to reserve and directly pay the "marvelous" www.acmefrozenpasta.com domain. From now on, three people contemporaneously work to build the site up. The first one is John, a technician, which analyzes what kind of computer would be better to purchase. Instead again

Jane takes care of making some phone calls to different web companies to collect the prices for hosting services. Finally Jack, the web designer of the ACME Co., starts planning and implementing the site structure. After some days, John has already chosen a proper configuration for the server and received back from different computer companies their estimates. Now he is in the phase of finalizing the report

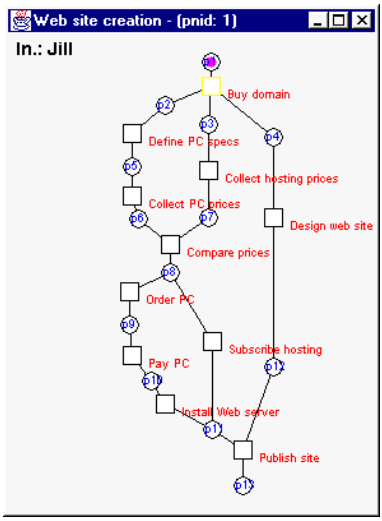

Fig. 5. The graphical representation of the model of the work process handling the creation of a new web site

summarizing the prices. However, while reviewing his report and just before forwarding the document to Jill, by chance he discovered that a new and probably cheaper server configuration could be used. What a pity; all previously collected prices are no longer meaningful because the configuration has to be changed. With a subtle sense of frustration, John undoes his configuring activity (exploiting a backward linear jump) and later redoes it so the collection of the estimates can be also made again. "Hum, at least this second round of estimate collection is easier to do than the first one.": John thinks. In fact, he can exploit the history recorded by MILANO and re-send the same messages requesting an estimate to the various companies just modifying the attached document with the new document containing the details of the right configuration. After getting all the estimates, both for a dedicated computer and a hosting case, a choice has to be made and it is up to Jill comparing prices and choosing the most valuable and affordable solution. In particular, Jill chooses a special offer made by the Computer-Sellers Inc.: they sell a limited number of web servers, totally compliant to John's specifications, for a very reasonable price. Therefore Jane, receiving the outcomes of Jill's decision, takes care of ordering the server. Unfortunately enough, after just a few hours a phone call arrives to Jane saying that no more web servers are available for that price. In fact, it was a limited offer. At this point Jane undo (again!) the John's activity of collection of prices (exploiting a *revised* backward linear jump). In fact, the above-mentioned

offer within his report is no longer valid. As a consequence of the Jane undo, Jill's decision and the order of the PC are canceled too. John makes just a small revision to his report (there is no need to collect estimates again since just a few hours have passed since they were asked) and completes its task. After that, Jill evaluates again the prices and actually hosting seems now a more valuable solution and therefore the previous choice is now changed. Hosting is subscribed with a well-know company at a reasonable price and, when Jack completes the design activity the site can be published, opened and the online advertising campaign can be started.

6 Conclusion

Discussing the undo solution of the MILANO WfMS prototype, we read the work by F. Leymann on irreversible actions and compensation spheres [34, 35]. The latter moved us to think that irreversibility of activities in a business process is a matter meriting a closer look if we want that a business process is robust. Robust in the sense that it is able to support undo and redo during its execution without loss of control. Extending the insight of Leymann, we can recognize that any activity has some inherently irreversible features (consumption of time and resources, market transactions with external actors, etc.), making undo *impossible* if its irreversibility is not compensated in a clearly defined way. This means that any business process cannot be robust, if all its activities are not coupled with their compensation spheres, so that their irreversibility becomes ineffective. This can be considered a basic guideline for robust business process design, indicating as a general principle, that every activity must be designed in such a way to be compensated, when exceptions occur. The undo strategy sketched above for database and information system transactions is in accordance with this rule. As another example, deciding that a computer bought for a project, if no longer needed within the project, must be restituted to the purchasing office, that will destine it to another project and/or office, is also a way to make the process robust, since canceled activities do not leave uncommitted resources in an organization.

The undo solution for WfMSs we have sketched in this paper is from our viewpoint an important piece allowing to link effectively WfMSs with corporate information systems. Up to now, WfMSs and information systems have been conceived, designed and developed as loosely coupled systems, where the former ones deal with documents while the second ones manage data. But, as we tried to explain in the previous pages, this separation does not work in real cases and would restrict the potential of WfMS technology so that it could not handle relevant processes in the life of an organization. The completion of the integration between WfMSs and information systems that a well designed and efficient undo realizes offers to WfMS technology a strong opportunity for gaining new users and application areas.

Solving the undo problem for WfMSs, we have been forced to reconsider the architectural issues regarding the integration between the former and information systems. In the Cooperative Information System manifesto ([36]) one of the authors and many colleagues have proposed a comprehensive framework proposing a new approach where cooperative applications and legacy systems are considered together. Let us recall briefly the main concepts proposed in the Manifesto, from the workflow

viewpoint. Cooperative Information Systems [37] are characterized by a three-faceted architecture: system, organizational and group collaboration facets (see [36] and [38]). From the system point of view, a WfMS must focus on the coordination among different activities and leaving to the other components the duty of performing them while avoiding to introduce any constraint to the interaction with other systems. From the organizational point of view, the WfMS must support and make visible the workflows: defining part of the rules, roles and procedures characterizing the company and/or institution where it is used; offering to its users the automation of the routine tasks; helping them to deal with exceptional situations and breakdowns; providing the distribution of decision tasks among the users (in accordance with their roles); simulating their performances under diverse resource allocations; giving a full support for the change of the workflows checking their consistency; and, finally, enacting changes in the ongoing workflow instances. Lastly, from the group collaboration point of view, it is an artifact within the workspace of their users supporting their performances; for instance, making the workflow transparent when they are already capable to do the requested activity; making the workflow visible when they need help in understanding what to do as well as when a breakdown requires them to negotiate a new cooperative work arrangement.)

In this framework it seems that the undo for a WfMS can be considered as a typical module at the interface between the group and system facets ([36], [38]).

Acknowledgements. The authors thank the anonymous reviewers who gave useful suggestions for improving the first version of this paper.

References

1. Schmidt, K., Bannon, L.: Taking CSCW Seriously: Supporting Articulation Work. Computer Supported Cooperative Work (CSCW). An International Journal, Vol. 1, Nos. 1–2 (1992) 7–40
2. Schmidt, K.: Of maps and scripts: the status of formal constructs in cooperative work. In S.C. Hayne and W. Prinz (eds.): GROUP'97. Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work. Phoenix, AR, November 16–19. ACM Press, New York, NY (1997) 138–147
3. Koulopoulos, T.M.: The Workflow Imperative. Van Nostrand Reinhold, New York, NY (1995)
4. Schael, T.: Workflow Management Systems for Process Organizations. 2nd Edition. Lecture Notes in Computer Science, Vol. 1096. Springer-Verlag, Berlin, Germany (1998) 142–153
5. White, T.E., Fischer, L. (eds.): The Workflow Paradigm. Future Strategies, Alameda, CA (1994)
6. Agostini, A., De Michelis, G.: Improving Flexibility of Workflow Management Systems. In W. van der Aalst, J. Desel, A. Oberweis (eds.): Business Process Management: Models, Techniques, and Empirical Studies, LNCS 1806. Springer-Verlag, Berlin Heidelberg New York (2000) 218–234
7. Agostini, A., De Michelis, G.: A light workflow management system using simple process models. In Computer Supported Cooperative Work: The Journal of Collaborative Computing, Kluwer Academic Publishers, Vol. 9, No. 3–4 (2000) 335–363

8. Berlage, T.: A Selective Undo Mechanism for Graphical User Interfaces Based on Command Objects. ACM Transactions on Computer-Human Interaction, Vol. 1, No. 3 (1994) 269–294
9. Meng, C., Yasue, M., Imamiya, A., Mao, X.: Visualizing histories for selective Undo and Redo. In Proceedings of APCHI98, Hayama, Japan (1998) 459–464
10. Sun, C.: Undo any operation at any time in group editors. Proceedings of ACM 2000 Conference on Computer Supported Cooperative Work, December, Philadelphia, Pennsylvania (2000)
11. Wang, X., Bu, J., Chen, C.: Group editing algorithms: Achieving undo in bitmap-based collaborative graphics editing systems. In CSCW'02. Proceedings of the conference on Computer Supported Cooperative Work, New Orleans, Louisiana (2002) 68–76
12. Ressel, M., Gunzenhäusler, R.: Reducing the Problems of Group Undo. ACM Digital Library, Proceedings of Group'99. Phoenix, AR. ACM Press, New York, NY (1999) 131–139
13. Borgida, A., Murata T.: Tolerating Exceptions in Workflows: a Unified Framework for Data and Processes. In Proceedings of the International Joint Conference on Work Activities, Coordination and Collaboration, WACC'99, San Francisco, CA. ACM Press (1999) 59–68
14. Casati, F., Ceri, S., Paraboschi S., Pozzi, G.: Specification and Implementation of Exceptions in Workflow Management Systems. ACM Transactions on Computer-Human Interaction, Vol. 24, No. 3 (1999) 405–451
15. Das, S., Kochut, K., Miller, J. Sheth, A. Worah D.: ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR2. (1997) Technical Report UGA-CS-TR-97-001, Department of Computer Science, University of Georgia
16. Ellis, C. A., Keddara, K., Rozenberg G.: Dynamic Change within Workflow Systems. In N. Comstock and C. Ellis (eds.). In Proceedings of the Conference on Organizational Computing Systems (COOCS'95). Milpitas, CA, August 13–16, 1995. ACM Press, New York (1995) 10–21
17. Kammer, P.J., Bolcer, G.A., Taylor, R.N., Bergman, M.: Techniques for Supporting Dynamic and Adaptive Workflow. Computer Supported Cooperative Work. The Journal of Collaborative Computing. Kluwer Academic Publishers, Vol. 9, Nos. 3–4 (2000) 269–292
18. Kiepuszewski, B., Muhlberg, R., Orlowska, M.E.: Flowback: Providing Backward Recovery for Workflow Management Systems. Proceedings of the International Conference on Management of Data, ACM SIGMOD (1998)
19. Miller, J.A., Palaniswami, D., Sheth, A.P., Kochut, K.J.: WebWork: Meteor2's Web-based Workflow Management System. Journal of Intelligent Information Systems, Vol. 10, No. 2 (1998) 185–215
20. Miller, J.A., Sheth, A.P., Kochut, K.J., Luo, M.: Recovery Issues in Web-Based Workflow. LSDIS Lab, Computer Science Department The University of Georgia, Athens, GA 30602–7404 (1997)
21. Weissenfelds, J., Wodtke, D., Weikum, G., Kotz Dittrich, A.: The Mentor Architecture for Enterprise-wide Workflow Management. Lecture Notes on Computer Science: Advances in Workflow Management Systems and Interoperability, Springer-Verlag, Berlin Heidelberg New York (1997)
22. Edwards, K., Igarashi, T., LaMarca, A., Mynatt E.: A temporal model for multi-level undo and redo. ACM Transactions on Computer-Human Interaction, Vol. 2, No. 2 (2000) 31–40
23. Prakash, A., Knister, M.J.: A Framework for Undoing Actions in Collaborative Systems. ACM Transactions on Computer-Human Interaction, Vol. 1, No. 4, December 1994, (1994) 295–330
24. Zhou, C., Imamiya, A.: Object-based nonlinear undo model. Proceedings of the COMPSAC '97 - 21st International Computer Software and Applications Conference (1997) 50–55

25. Verhofstad, J.S.M.: Recovery Techniques For Database Systems. In ACM Computing Surveys, Vol. 10, No. 2 (1978) 109–123
26. Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P.: ARIES: A Transaction Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. ACM Transactions on Database Systems, Vol. 17, No. 1 (1992) 94–182
27. Kuo, D.: Model and Verification of a Data Manager Based on ARIES. ACM Transactions on Database Systems, Vol. 21, No. 4, (1996) 427–479
28. Suchman, L.: Plans and Situated Actions. The problem of human-machine communication. Cambridge University Press, Cambridge, UK (1987)
29. Winograd, T., Flores, F.: Understanding Computer and Cognition: A New Foundation for Design. Ablex Publishing Corp., Norwood, NJ (1986)
30. Agostini, A., De Michelis, G., Grasso, M.A.: Rethinking CSCW systems: the architecture of MILANO. In J. Hughes et al. (eds.): ECSCW'97. Proceedings of the Fifth European Conference on Computer Supported Cooperative Work, Lancaster, UK, September 7–11, 1997. Kluwer Academic Publishers, Dordrecht, The Netherlands (1997) 33–48
31. Rozenberg, G., Engelfriet, J.: Elementary Net Systems. In W. Reisig and G. Rozenberg (eds.): Lectures on Petri Nets I: Basic Models. Lecture Notes on Computer Science, Vol. 1491. Springer-Verlag, Berlin Heidelberg New York (1998) 12–121
32. Jüngel, M., Kindler, E., Weber, M.: Towards a Generic Interchange Format for Petri Nets – Position Paper. Humboklt-Universität zu Berlin, Institut für Informatik, D-10099 Berlin, Germany (2000). Proceedings of Meeting on XML/SGML based Interchange Formats for Petri Nets, 2000.
33. Jüngel, M., Kindler, E., Weber, M.: The Petri Net Markup Language. Humboklt-Universität zu Berlin, Institut für Informatik, D-10099 Berlin, Germany (2000)
34. Leymann, F.: Supporting Business Transactions via Partial Backward Recovery in Workflow Management Systems. Proceedings of BTW '95, Springer-Verlag, Berlin Heidelberg New York (1995)
35. Leymann, F., Roller, D.: Workflow-based Applications. *IBM Systems Journal*, Vol. 36, No. 1 (1997) 102–123
36. De Michelis, G., Dubois, E., Jarke, M. Matthes, F., Mylopoulos, J., Papazoglou, M.P., Pohl, K., Schmidt, J., Woo, C., Yu, E.: Cooperative Information Systems: A Manifesto. In M. P. Papazoglou and G. Schlageter (eds.): Cooperative Information Systems: Trends & Directions, Academic-Press New York (1998) 315–363
37. Papazoglou, M.P., Schlageter, G. (eds.): Cooperative Information Systems: Trends & Directions. Academic-Press New York (1998)
38. De Michelis, G., Dubois, E., Jarke, M., Matthes, F., Mylopoulos, J., Schmidt, J. W., Woo, C., Yu E.: A Three-Faceted View of Information Systems: The Challenge of Change. Communications of the ACM, Vol. 41, No. 12 (1998) 64–70
39. De Michelis, G., Dubois, E., Jarke, M. Matthes, F., Mylopoulos, J., Papazoglou, M.P., Pohl, K., Schmidt, J., Woo, C., Yu, E.: Cooperative Information Systems: A Manifesto. In M. P. Papazoglou and G. Schlageter (eds.): Cooperative Information Systems: Trends & Directions, Academic-Press New York (1998) 315–363

A Top-Down Petri Net-Based Approach for Dynamic Workflow Modeling*

Piotr Chrząstowski-Wachtel[1**], Boualem Benatallah[2], Rachid Hamadi[2], Milton O'Dell[3], and Adi Susanto[2]

[1] Institute of Informatics, Warsaw University
Banacha 2, PL 02-097 Warszawa, Poland
pch@mimuw.edu.pl
[2] School of Computer Science and Engineering
The University of New South Wales, Sydney NSW 2052, Australia
{boualem,rhamadi,adis}@cse.unsw.edu.au
[3] Justwin Technologies Pty Ltd
7-9 West Street Suite I.20, Level 1, North Sydney NSW 2060, Australia
modell@justwin.com

Abstract. A top-down approach for workflow design is proposed in the framework of Petri net theory. Simple but powerful refinement rules are proposed that guarantee soundness of the resulting workflow nets. The refinement process supports the definition of regions, which are parts of the workflow that correspond to logistically related items. Exception handlers can be associated to regions. Defining regions helps determining the impact areas of the unexpected events during workflow execution.

1 Introduction

Workflow management systems are used for controlling the execution of business processes. These processes typically consist of multiple activities, which have to be performed in a valid sequence [1,2]. Dynamic workflows deal with changes during the workflow execution. Some of them are unpredictable, and it is hence necessary to allow flexible modeling and run-time maintenance. The main problem here is to restrict the impact area of unexpected exceptions. Recovery regions [3] form a partition of the workflow into areas to which the reaction for the exceptions is limited to. A flat structure of regions was considered in [3]. Since exceptions are of different importance, it is reasonable to make the reactions structured so that we achieve two goals. First, only the minimum part of the workflow is affected. Second, provide a mechanism for handling exceptions at

* This work is partially supported by an ARC SPIRT grant "Managing Changes in Dynamic Workflow Environments" between UNSW, QUT, and Justwin Technologies and by an internal research grant No. BW/ALG/01/2002 of PJWSTK financially supported by KBN in Poland.

** Also with Polish-Japanese Institute of Information Technology, Koszykowa 86, PL 02-008 Warszawa, Poland.

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 336–353, 2003.

different levels of the design that often reflect different levels of management. Certain recovery decisions will be limited to appropriate levels of management. If the design is made accordingly, the levels will be determined at design phase.

A hierarchical Petri net approach will be presented in this paper. The key idea is to design a workflow in a top-down manner, starting from a single place and performing refinements using a given set of rules. The refinement rules guarantee that the workflow will enjoy the desired soundness property [4]. The refinement rules are chosen to be simple enough to use and powerful enough to represent many common workflow patterns identified in [5].

The paper is structured as follows. Section 2 describes the design of workflows using basic refinement rules. Section 3 deals with non-refinement rules, i.e., communication and synchronization, as well as with soundness property. The recovery regions are presented in Sect. 4. Section 5 describes the *HiWorD* (HIerarchical WORkflow Design) tool. Finally, Sect. 6 concludes the paper.

2 Designing Workflows by Refinement

Workflows may represent significantly large business processes, having hundreds or thousands of actions to be performed upon completion. As usual, when dealing with complicated processes, it is important to follow a structured design model.

Our approach will concentrate on building a hierarchical workflow model based on Petri nets, a model used in concurrency theory, which has proven to support efficiently the hierarchization concept. Since the structure of the top-level tasks in most workflows is not too complicated, we decide to split the design process into two phases. First, the design of the overall workflow structure, which will be restricted by some rules allowing to define tasks in a safe way on top levels. Then specifying the details of the task description at a lower level, allowing some primitives, which will make the design safe, in the sense that it will not cause deadlocks [6,7].

It is tempting to use simpler models as they are easy to analyze and implement, but it is undesirable to limit the expressive power of the model. Our approach aims at making the workflow structured as in structured programming languages. A method similar to procedural encapsulation together with the idea of procedure nesting will be used for determining the workflow integral parts.

At some stage we let the designer define *regions*. They represent the encapsulation concept in the workflow design. We can think of them as *milestones* of workflow execution, i.e., the completion of region execution should result in reaching a state of execution, that typically once completed will not be re-done. Since our approach aims at unexpected exceptions, we associate with each region exception handlers that restore the workflow execution within a region once an exception is raised. Due to the nested structure of regions, different recovery levels can be described. A good design will reflect the management structure and support the decisions to be taken at an appropriate level.

We will illustrate the approach using the stepwise refinement method. Places in Petri net are considered not static states, but rather as representing the state

of execution of a part of the workflow. Beginning with a single place, being the coarsest view of the workflow description (i.e., the root of the refinement tree), the basic transformations allow to model the sequences of actions, the choice between two or more tasks, and the parallel split. The five basic refinement transformations are shown in Fig. 1.

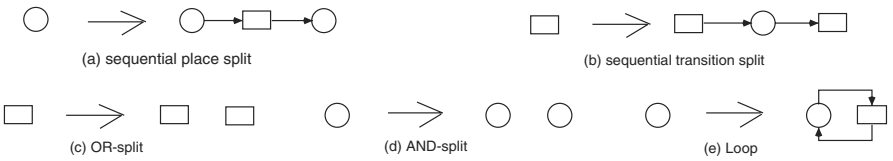

Fig. 1. Basic Refinement Rules

For all the rules displayed in Fig. 1, except the first one, we require that they can be applied only if there is at least one input and at least one output arc associated with the node to make the transformation valid. Moreover, we presume that all the input arcs of the node are copied to all the resulting entry nodes, and that analogous rule applies to the output arcs.

2.1 Workflow Refinement Rules

The first two pairs are dual, while the Loop-split rule does not have its dual counterpart. Observe that we do not want to introduce a dual rule of refining a transition, since we would face the problem of marking the input place. We would like to mark the whole workflow net (WF net) with only one token in the input place. This would mean, that the looping place would block its transition from being ever executed.

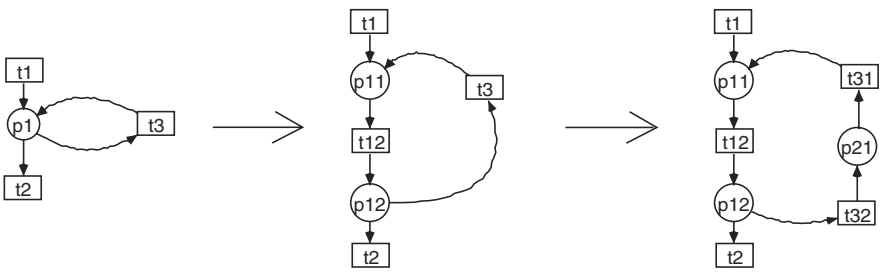

Fig. 2. Further Loop Refinement

The Loop-split rule allows us to construct loops having only one input and one output, both being places. When performing a sequential split, the place

that the loop was created from will be refined and the input and the output of the loop will be separated as in Fig. 2. We can clearly see, that when we refine the initial loop place, the actions that occur will be performed in any loop turn (action t_{12} in this case). When we refine the loop transition, we are in fact designing actions that should be performed between the failed loop-exit-tests and re-doing the loop again (t_{31}, t_{32}). So they serve for the re-initialization of the loop.

It should be noted that not all WF nets can be constructed using the proposed rules. The following theorem allows to specify most important features of such nets.

Theorem 1. *If we start with a single place and apply the above transformations under the constraint that, except for the sequential place split, all the transformations may be applied only if a node has at least one input and at least one output arc, then the resulting WF net* $\mathcal{N} = \langle P, T, \rightarrow \rangle$ *enjoys the following properties:*

1. *There is exactly one place* $p_{in} \in P$ *such that* $\forall x \in P \cup T \; p_{in} \rightarrow^* x$. *We call* p_{in} *the workflow input,*
2. *There is exactly one place* $p_{out} \in P$ *such that* $\forall x \in P \cup T \; x \rightarrow^* p_{out}$. *We call* p_{out} *the workflow output,*
3. *The number of incoming arcs in the WF net is equal to the number of outgoing arcs,*
4. *The WF net* $\mathcal{N}$ *is a free-choice net, and*
5. *The WF net* $\mathcal{N}^{-1}$ *resulting by reversing the arcs is a WF net, which is obtainable by the same set of refinements.*

Proof. The induction on the number of transformations made completes the proof of each of these properties. □

Recall, that a Petri net is a free-choice net if and only if for every two transitions, if they share any input place, then all their inputs are the same [8]. This condition guarantees that either both transitions are enabled or none of them is enabled under any marking.

2.2 On the Expressiveness of Refinement Rules

The proposed transformations are ready to express quite a variety of typical situations that occur in workflow design. However, there are some restrictions, that is, not every Petri net with one input and one output place (WF nets are assumed to have this property) can be generated using these rules.

Consider for instance the WF nets depicted in Fig. 3. These nets cannot be obtained by applying the rules given in Fig. 1. The net in Fig. 3(a) cannot be obtained because the total number of arcs being inputs to transitions is 4 and being outputs is 3, hence violating Theorem 1(3). The net in Fig. 3(b) cannot be obtained because it is not a free-choice net, thus violating Theorem 1(4).

In fact, to describe intuitively what is the main restriction on the class of WF nets obtained by the above rules, it should be noted that, although both

parallel splits and synchronizations are allowed, the following constraint must be satisfied: *what has been split must be synchronized later*. Similarly *what has constituted a simple exclusive choice must be later merged*. Recall, that although the net of Fig. 3(a) violates this principle (an exclusive choice if followed by a parallel merge), the net of Fig. 3(b) one does not. There is a triple OR-split at the top and a triple OR-join at the bottom, as well as three AND-splits, each of them followed by an AND-join. The problem with this net is that, although the numbers of splits and joins of the same type are the same, the OR- and AND-splits and joins are shuffled so that we obtain a sequence: OR-split followed by AND-split followed by OR-join followed by AND-join. In our model, nesting is allowed but not shuffling.

The first net represents a design structure that we want to avoid since it causes a deadlock. Similar difficulties would be faced in the dual case of a parallel split followed by a merge, resulting in a generation of a new token. The latter situation is at first glance less unsafe. However, the second net represents a kind of normal situation. Since we don't want to rule out this kind of behavior, we will consider such transformations possible on a lower level that will allow us to represent arrangements of this kind.

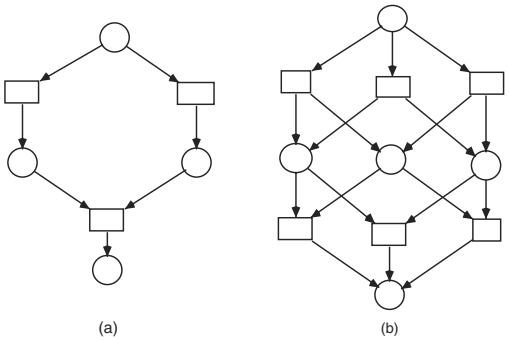

Fig. 3. Nets that can not be obtained by using the basic refinement rules

The proposed rules correspond to the first five patterns[1] identified by van der Aalst et al. [5], but with the restriction that the parallel split must be followed by the parallel synchronization, while the exclusive choice must be followed by a simple merge. This restriction is considered not to be a deficiency. Rather it introduces a sound constraint that prevents us from making obvious design errors. At the management level the above rules will suffice for a large amount of cases. What we gain is the guarantee that the resulting WF net is correct with respect to the *soundness conditions* as identified in [4].

[1] http://www.tm.tue.nl/it/research/patterns/.

Definition 1. *A sound WF net is a Petri net with places p_{in} and p_{out} that satisfies the following conditions:*

- *For each token put in p_{in}, one and only one token eventually appears in p_{out},*
- *When the token appears in p_{out}, all other places are empty, and*
- *For each transition (i.e, task), it is possible to move from the initial state to a state in which this transition is enabled.*

Theorem 2. *If we start with a WF net consisting of a single place with no transitions, and perform only the proposed transformations, then the resulting WF net is sound.*

Proof. The induction on the number of refinements performed completes the proof. □

2.3 Refinement Tree

In the next sections we will provide the rules that allow us to construct basic WF nets in order to support elaborated patterns like communication or synchronization between concurrent parts of the net. In order to do so, it is desirable to define the notion of concurrency in such nets precisely, and to have a fast algorithm for determining if two nodes are truly concurrent. For this purpose, we use the notion of *refinement tree*, which will be introduced in this section. We show that the refinement tree will be defined in a unique way for each net obtained by a sequence of basic refinements.

Note that, after applying several times the transformation rules, we obtain a tree structure reflecting the order in which the transformations were made. The places and transitions of the net will label the leaves of the tree, while the internal nodes will be labeled by the transformation names. Let us assume that the tree is ordered, so that we can identify the role of each node in case of non-symmetrical (sequential) transformations. The root of the tree is initially labeled by the initial place we start our design from. Since each transformation rule is of the form $\langle \text{node} \rangle ::= \langle \text{net} \rangle$, when we apply any of these rules, the node from the left hand side of the transformation description already exists in the tree being its leaf. We then create its children labeling them by the resulting nodes, and marking the new parent node by the transformation name. The tree is uniquely determined by the order in which the transformations were made.

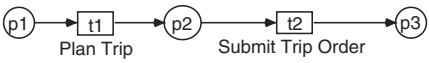

Fig. 4. A Sequence of Two Transitions

Since the refinement tree will be later used for characterization of sound transformations, which we are going yet to introduce, it would be desirable to

associate a unique tree to every WF net that resulted as a sequence of canonical refinements. So far this is not the case. To illustrate that let us consider the WF net of Fig. 4, for which three non-isomorphic trees can be created, as depicted in Fig. 5.

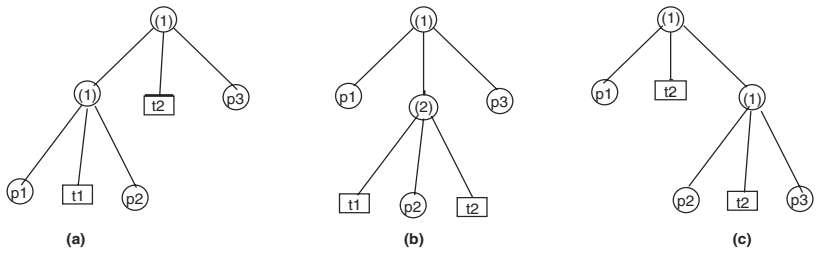

Fig. 5. Three Non-isomorphic Trees for the Net from Fig. 4

The refinement trees always have internal nodes labeled by the refinement rules, and the leaf nodes by the transitions and places. The trees are ordered, so that from such trees we are able to reconstruct the refinement process. The reason for which we cannot make the opposite (draw the tree when looking at the net) is that we are too accurate in performing sequences of refinements of the same kind. We distinguish too many details. Instead, we would prefer to look at such sequences of refinements as one-level refinements. In case of the example of Fig. 4, there should be one place sequential split node with five children: $p1$, $t1$, $p2$, $t2$, and $p3$, hence having only two levels instead of three.

We would like to contract the tree in case a parent and all its child nodes are labeled by the same transformation. In this case we would make all such children to become siblings of the parent by shifting them one level up. This operation reflects the extension of the rules to be n-ary instead of binary. For instance, we could define n-parallel split, n-parallel choice, and so forth. As both place sequence split and transition sequence split are reflecting a very similar kind of refinement, we will not distinguish between them, and treat them as one kind of refinement. The tree resulting from the original tree by such contraction will be called the *refinement tree*.

The height of the tree forms a basis for defining complexity of the workflow. From this point of view the workflow does not become more complicated if we add one more node to a sequence, but it does if we perform a parallel split in a sequence or sequentialize something that has been previously split.

To make the refinement tree canonical, hence unique for a given refinement net, we must define an equivalence relation on the refinement trees, so that non important differences in the construction will not lead to distinguishing the resulting two trees as different. The main problem is the importance of the order of children nodes depending on the kind of the father node. It turns out that if the sequential refinements take place, the order of children is important, as we

wish to distinguish between the first, and the last among the 3 resulting nodes, which are of the same kind and could be easily confused if the tree was unordered. At the same time, the other 3 refinement rules make the order irrelevant: there is no reason why we would have to say which of the nodes is first and which is the second child. They are identical at this stage. Let us define the equivalence relation on refinement tree recursively:

Definition 2. *Two trees T_1 with root v_1 and T_2 with root v_2 are equivalent iff one of the following conditions holds*

- *T_1 and T_2 are one-node trees and the types of the nodes is the same.*
- *If v_1 and v_2 are labeled by the sequential refinements, then the number of children of v_1 is the same as the number of children of v_2, and the corresponding children are equivalent.*
- *If v_1 and v_2 are labeled by OR-split, AND-split, or Loop-split, then there is a bijection between equivalent children of v_1 and v_2.*

It turns out that whenever different orders of refinements lead to the same net, then the contracted trees are identical for both of them.

Theorem 3. *If the WF net $\mathcal{N}$ resulted from a sequence of considered refinements then, no matter in which order they were made, the resulting refinement tree is uniquely determined.*

Proof. The proof will be an induction on the number of internal nodes n (each internal node represents one refinement). The theorem holds trivially for $n = 0$. Assume that $n > 0$ and for every $k < n$ all the nets with at most k internal nodes have isomorphic trees. Consider a WF net $\mathcal{N}$ having n nodes with two refinement trees of $\mathcal{N}$ representing different orders which led to the construction of $\mathcal{N}$. Each of the trees results as a sequence of refinements. Consider the last refinement made in the construction of T_1 and the last refinement of T_2. Let T_1' and T_2' be the trees before the last refinements made. Consider two cases:
Case 1. The areas touched by the last considered refinements are disjoint (i.e. none of them has a common node with the other one to share). Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be the nets corresponding to T_1' and T_2'. Let $\mathcal{N}_1'$ be the net resulting from $\mathcal{N}_1$ by undoing the possible refinement which was the last in $\mathcal{N}_2$. Similarly let $\mathcal{N}_2'$ be the net resulting from $\mathcal{N}_2$ by undoing the possible refinement which was the last in $\mathcal{N}_1$. Undoing these refinements is possible thanks to the assumption about the areas being disjoint. Now we got two isomorphic nets, and since the number of nodes is smaller than n, their corresponding refinement trees are also isomorphic. Now the only way to get the initial net $\mathcal{N}$ is to redo the two withdrawn refinements obtaining two isomorphic trees.
Case 2. The last refinement made in the construction of T_1 has a node in common with the last refinement made in the construction of T_2. In this case it is enough to verify that either the two last refinements were done in the same level, and both were of the same type (in particular sequential refinements of any type), in which case the order they were made is irrelevant, or they were indeed

refinement of different types, in which case we come to the contradiction, because no pair of different refinements is commutative. And we come to a contradiction: we cannot obtain the same (isomorphic) net. By not being commutative we mean, that starting from one node and performing any pair of different refinements (the only way to get the areas intersected) we always obtain non-isomorphic nets. This property can be checked manually, as there is a finite number of such pairs. □

Each WF net hence produces a uniquely determined (up to isomorphism) *refinement tree*. We will refer to this tree later, when defining new constructs, which allow us to have synchronization and communication between parallel parts. The tree will be used to determine the parallel parts of the WF net.

3 Non-refinement Rules

The refinement rules proposed so far are not sufficient for all situations that can occur during workflow design. We claim that they suffice for defining the region structure. Such structure is usually simpler than the detailed design, which may require more elaborate rules.

The key idea of this approach is to find a set of refinement rules that is sufficient to design typical workflow behavior. Workflows constructed using such rules: (i) enjoy all desired properties, such as soundness as mentioned in [7], (ii) have a structure easy to understand and maintain that reflects the management structure, and (iii) express most typical workflow patterns.

The structural design we have proposed so far is simple and quite powerful, being able to represent many real life situations. However, there exist patterns, which can not be obtained in a hierarchical way. The key problem is the interaction between concurrently enabled activities. We distinguish such patterns, as they occur in many workflow designs. All these patterns do not follow the refinement schema, where a single node is being transformed into a more complicated net. The proposed patterns involve more than one node at a time, and build a net structure between them according to some restrictions that guarantee soundness. The soundness property is the one we would like to keep, even at a price of some limitations in design freedom.

In fact all the proposed patterns will have something in common. They will join in certain ways different parts of the WF net by introducing new transitions or places serving as bridges. The important thing is to restrict such auxiliary constructs to connect only parallel parts of the net, in order to avoid the situation where an active part of the net would interact with an inactive one. Another restriction is that such constructs should not introduce loops, as they could easily cause deadlock.

In the following, we denote by place-type nodes in the refinement tree places, sequential place splits, AND-splits, and Loop-splits, while transition-type nodes stand for sequential transition splits and OR-splits, as identified by the canonical refinements (see Fig. 1).

Definition 3. *Two nodes x_1 and x_2 (places or transitions) in a WF net obtained as a result of the canonical refinements lie in parallel threads iff in the refinement tree the only nodes on the path from x_1 to x_2 are the ones labeled by AND-split, sequential splits, or Loop-splits, but in this last case only followed by a place-type node.*

Therefore, there may be neither OR-splits nor Loop-splits followed by a transition-type node (transition, sequential transition split, or OR-split) on the path joining two nodes that we define being on parallel threads.

The notion of parallel threads will be helpful in defining sound patterns which allow us to join different parts of the WF net in a non-refinement manner. The key idea is that when two nodes are in parallel threads, then in all complete runs (from the token on p_{in} to the marking with a token on p_{out}) either both of them occur (place holds a token or transition fires) or none. Moreover, for all pairs of such nodes there exist such runs, in which each of them precedes the other one.

Theorem 4. *If a Loop-split followed by a transition-type node or an OR-split occurs on the path from x_1 to x_2 in the refinement tree, then there exists a complete run such that one and only one of x_1 and x_2 occurs on this run.*

Proof. When an OR-split occurs on the path from x_1 to x_2, then both of the nodes lie in different choices of one decision. Since the net is free-choice, when we make the choice in favour of one node, the other one cannot be activated. When a Loop-split followed by a transition-type node occurs on the path from x_1 to x_2 in the refinement tree, it means that x_2 occurred as a result of refining the transition, that closes the loop backwards. Any attempt to enforce the occurrence of this backward transition is hopeless, since we have free-choice decision whether to continue the loop or to leave it. Note, that the refinement of the Loop-place into a sequence produces part of the loop that will always be performed, so there is no need to exclude the Loop-splits on the path in general. □

The reverse statement is also valid as stated by the following result.

Theorem 5. *If neither a Loop-split followed by a transition-type node nor an OR-split occurs on the path from x_1 to x_2 in the refinement tree, then either both x_1 and x_2 or none of them occur in all complete runs.*

Proof. The sequential splits just carry tokens along the nodes that result after the split, so indeed a sequential split carries no danger that only one of x_1 or x_2 will occur in the complete run. Similarly the AND-split activates both places simultaneously. The Loop-split pattern followed by a place-type node plays the role of a part of the loop which will always be executed once a token occurs in the initial place after the refinement. So the entire sub-net that is the result of further refinement preserves the desired property. Induction completes the proof. □

These two theorems will form the basis for formulating the conditions for sound non-refinement patterns, which connect existing nodes in the WF net.

3.1 Communication

The first of the non-refinement patterns is *communication*, where one agent wants to send a message to another, and both are concurrently running. We assume here that all messages are important and that the receiver will wait for the expected message until it arrives. This construct is somewhat of different nature and should be used with extreme care. We will propose a condition that will allow a safe introduction of a new communication place, creating a buffer for message passing without loosing soundness condition.

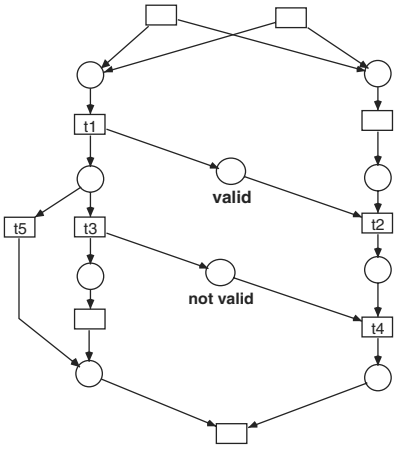

Fig. 6. Communication

In many cases, when the execution of parallel threads appears in a workflow, a communication should occur involving sending a message from one thread to another one. We assume that, the recipient of the message should wait until the message arrives. So we treat messages like signals that enforce synchronization between parallel parts of a workflow rather than information spread across the workflow diagram. In order to send a message, one needs to specify the sender and the receiver, being aware that they are indeed active agents. A deadlock can occur if the receiver is expecting a message from the sender that is not activated because, for instance, it was not chosen at an OR-split point.

At design phase, we will create the communication links by pointing at two partners of communication, but with an additional restriction. They should lie on parallel threads, and the introduced communication place should not close a cycle.

Consider the net example of Fig. 6. One can verify, that the transitions $t1$ and $t2$ lie on parallel threads. It would be perfectly valid to make a communication from $t1$ to $t2$, as well as the opposite way around. But we do not allow a communication to be performed between $t3$ and $t4$, because in the refinement

tree on the path leading from t_3 to t_4 there is an OR-split. In case a choice for $t5$ has been made in favor of $t3$, the transition $t4$ would be made dead. In fact the presence of an OR split on the path between two communicating agents is inherently bad, as the net is free choice and we can always make a wrong decision causing a deadlock or a trash token, which could possibly never be taken away.

Moreover, it should not be allowed to send a message to a node, which is our predecessor in the net, hence closing a cycle. But this will be excluded by the restriction we impose on the communication agents as defined below.

Definition 4. *Let t_1 and t_2 be two transitions. A place p joining t_1 and t_2 is a communication place iff the following conditions are satisfied:*

- *t_1 and t_2 lie on parallel threads, and*
- *Introducing the communication place does not close a cycle.*

To preserve soundness, both conditions should be satisfied. In case t_1 and t_2 are not in parallel threads, a scenario can be created in which soundness is lost.

3.2 Synchronization

It is often the case that two parallel threads should synchronize their activities before they are completed. We distinguish here between two cases: *symmetric synchronization* and *asymmetric synchronization*. The first one occurs when there are checkpoints in each of them that must be simultaneously reached. The asymmetric one occurs when one of them is privileged: it does not need to wait for the other one to proceed, while the other one can not go across the checkpoint if the first one has not reached it.

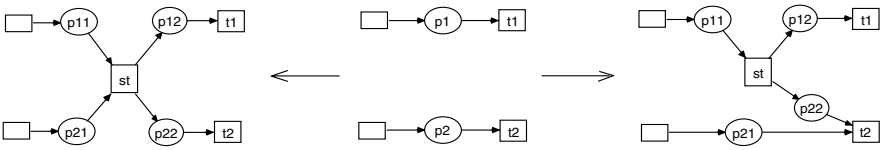

Fig. 7. Synchronization

Symmetric Synchronization. The symmetric synchronization can be obtained by the pattern depicted on Fig. 7(a). We are supposed to synchronize the parallel threads on places $p1$ and $p2$. Neither $t1$ nor $t2$ may fire unless the other one is enabled. In other words, we would like to enable both of them as soon as tokens appear on both $p1$ and $p2$. As a result of the synchronization, the synchronized places $p1$ and $p2$ are split and the synchronizing transition st fires as soon as tokens appear on their "left" halves, i.e., on $p11$ and $p21$. It is

assumed that, this transition fires without any delay as soon as it is enabled. As a result of firing st we obtain tokens on places $p12$ and $p22$. They enable transitions $t1$ and $t2$ as required.

Asymmetric Synchronization. The asymmetric synchronization can be obtained by the pattern depicted on Fig. 7(b). This time we want to let the upper thread proceed without waiting for the lower one, but the lower thread should wait until a token appears on $p1$ in the upper thread. This pattern, like the symmetric synchronization, should be made under the same condition as the communication pattern. In fact, communication involving the transition immediately preceding the synchronization place in the dominating thread could be an option for realizing this pattern.

Definition 5. *Let p_1 and p_2 be two places. A synchronization pattern may be added involving p_1 and p_2, as in Fig. 7, iff the following conditions are satisfied:*

- *p_1 and p_2 lie on parallel threads, and*
- *Introducing the synchronizing transition does not close a cycle.*

3.3 Soundness Property

All the introduced patterns have a similar condition: the only parts of a WF net that can interfere safely (without loosing soundness property) with each other by some additional constructs, are the ones that occur in parallel threads. Hence, the threads that were generated from an AND-split as a result of further refinement of places which were in the right-hand side of the refinement rule.

Theorem 6. *If we start with a single place and proceed with the canonical refinements as well as with the communication and synchronization patterns under the given restrictions, then the resulting WF net is sound.*

Proof. Let us concentrate on the communication pattern, as the proof for the synchronization patterns follows the same line of reasoning. Assume, that t_1 wants to send a message to t_2 involving communication pattern and both lie on parallel threads, and the introduced new place does not close a cycle. Theorem 4 says that either both transitions will be absent in a complete run (in this case soundness is trivially preserved) or both will be present. In the latter case the introduced place may enforce the order in which the transitions will be executed. If in all runs, in the original net, t_1 precedes t_2, then the same runs will be possible after introducing the communication place. If in all runs, in the original net, t_2 precedes t_1, then it means that, t_2 is a predecessor of t_2 in the net, hence introducing the communication place from t_1 to t_2 would close a cycle.
If in some runs t_1 precedes t_2 and in some other ones t_2 precedes t_1, then introducing the communication place will enforce the order in which they are executed, but will not cause a trash token or a deadlock. The token put on the communication place will be consumed by t_2. And the communication place cannot cause a deadlock, because it will withhold t_2 from firing until t_1 fires, and

since neither of them is a predecessor of the other one and the net is free choice, we can always execute the runs in which t_1 precedes t_2. □

Observe that, the conditions under which the communication and synchronization patterns were allowed are also necessary. The reader can verify, that if the involved nodes do not lie on parallel paths or the introduced net closes a cycle, then there is always a possibility of creating a scenario that causes deadlock or trash tokens.

4 Recovery Regions

We will now apply the proposed top-down approach to the definition of *recovery regions* [3]. We propose another idea, that is, to use the refinement process as a natural method to define regions.

Regions represent parts of a workflow. Region elements are related to each other and can be identified as a whole activity to be performed, clearly indicating its origin and end. We want regions to be workflows themselves, enjoying all the workflow properties. We also associate with regions exception handlers. Each exception handler will determine the compensation procedure and entry points, so that some amount of work done so far can be saved. We would like to restart the execution of a region at the latest possible entry point. In the sequel, we introduce the encapsulation mechanism, which will allow us to define recoveries in a structured way.

4.1 Design Overview

At any step of the refinement process, we can request a place to become a region. To each region several exception handlers can be assigned. Each such compensation procedure will affect the execution of the region in the following way. The entire region's interior will be reset, and tokens will be deposited on certain places associated with the recovery transition. By resetting the region's interior, we mean stopping all the activities being executed, removing all the tokens from the interior of the region, and performing some compensation procedures as needed. Depositing tokens means that we specify the entry points, from which the execution of the region will be resumed. This can save some work done so far, however quite often it can happen that a default reset, i.e., putting a token on the region input place, will be performed. The default reset means we will start executing the whole region from the beginning.

Once a place p is refined to a region, all the subnet resulting from p forms the region's interior. The subtree of a node, that was called a region, represents the interior of a region. The place, being the first one in the left-to-right in-order traversal of the subtree associated with a region, is the input place of the region, while the last one is the output place of the region. Note that any region is a valid WF net by itself.

Regions represent typically the milestones of a workflow execution. Once passed, they will not usually be redone. The transitions between the regions

play an auxiliary role of transferring the control from one or more regions to some other ones. However, sometimes an unexpected event may necessitate to move backward and forward between regions, as parts of workflow will have to be redone or skipped. For example, if a transport of produced garments is stolen or destroyed, we must go back to the production phase, but usually after the design phase. On the other hand, if during quality assessment, for instance, we require three independent checks, and only two are ready by the deadline, we can decide to skip waiting for the last one to arrive and go to the next phase which form another region. Such events require triggering a recovery transition which can cause, for instance, stopping all the activities that concern the stolen or destroyed goods and initiating the production phase in the first example, or informing the quality assessment third party about abandoning the need for the review and preparing the unit which is responsible for the collection of the reviews to discard anything that comes from it in the second example. Similar procedures could be taken at any level of the description since every region is a workflow by itself. The only difference is that the management level is different and the communication is more local.

4.2 Example: Ordering Flight Tickets

We give here an example of workflow representing ordering flight tickets over the Internet to illustrate the concepts introduced. A customer plans her/his trip by specifying the various stages of the overall journey. Then s/he sends this information together with the list of all participants of the trip and the information about the credit card to be charged for the ordered tickets to the travel agent and waits for the submission of the electronic tickets as well as the final itinerary before preparing for the trip by making other arrangements such as hotel booking and car renting. When the travel agent receives the customer's trip order, s/he will determine the legs for each of the stages, submits these legs together with the information about the credit card to be charged to the airline company, and waits for the confirmation of the flights. This information is completed into an itinerary and sent to the customer. When the airline receives the tickets order submitted by the agent, the requested seats will be checked and, if available, assigned to the corresponding participants. After that, the credit card will be charged and the confirmation of the flights sent. Finally, once the travel agent receives confirmation, the airline sends the electronic tickets by e-mail to the customer.

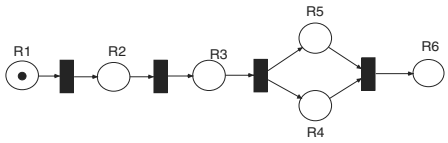

Fig. 8. Ordering Flight Tickets High-level Workflow

Starting from a single place and using a succession of refinement rules, the high-level workflow shown in Fig. 8 is obtained. Black rectangles stand for silent (or empty) transitions. By further refining places `R1` through `R6`, we obtain the flat workflow represented in Fig. 9. An example of regions associated within the workflow is also given in Fig. 9. Regions are depicted by dotted polygons, recovery transitions by dashed boxes, and recovery arcs (i.e., arcs linking regions to recovery transitions) by attaching two arrowheads instead of one to the end of the arcs. For presentation clarity, we restrict the detail of recovery transitions to region `R3` only. Within a region, for all other (unexpected) exceptions that do not match the expected exceptions, a default unexpected exception handler is raised. This can be, for instance, undoing the effects of the activities and resuming the execution of the region from the beginning. For region `R3` in Fig. 9, this is represented by the recovery transition `t3`.

Such decisions must be done at an appropriate level. If management levels are clearly identified then a natural workflow design decision would be to create regions in a way that reflects the management levels and associates certain privileges to trigger recoveries at each level. Consider a situation in which manager A, responsible for part α of the workflow, has a subordinate B, responsible for part β being a subpart of α. In this case A will be able to perform more general recoveries, involving parts of the workflow managed by B, because her/his recovery procedures can also reset the part β of the workflow. On the other hand, B can only handle exceptions for β without having permission to trigger recovery procedures outside β.

Such exception handling would require channels for communication ready for transmitting recovery decisions and a language clearly indicating the actions to be performed, since recoveries should follow some predefined patterns. Typically, the regions' structure will create ideal opportunities for the organization of message transmissions making them hierarchical.

5 Tool Support

To illustrate the viability of the approach presented in this paper, we have developed *HiWorD* (HIerarchical WORkflow Designer), a hierarchical Petri net tool using Java [9]. The tool has been successfully used to hierarchically design workflow scenarios in a safe and effective way from our project industrial partner Justwin Technologies Pty Ltd[2]. We plan to integrate the modeling tool with Justwin Workflow engine.

HiWorD is an editor tool that can be used to provide an efficient way to design and display workflows, by using the concepts of refinement rules and regions. In addition to traditional workflow modeling, *HiWorD* allows, starting from a single place, the refinement of places and transitions to create hierarchical workflows. The tool supports also the concept of region. A region is considered as a refined place in *HiWorD* for which one or several recovery transitions are added and each recovery transition is associated with an exception handler.

[2] http://www.justwin.com/.

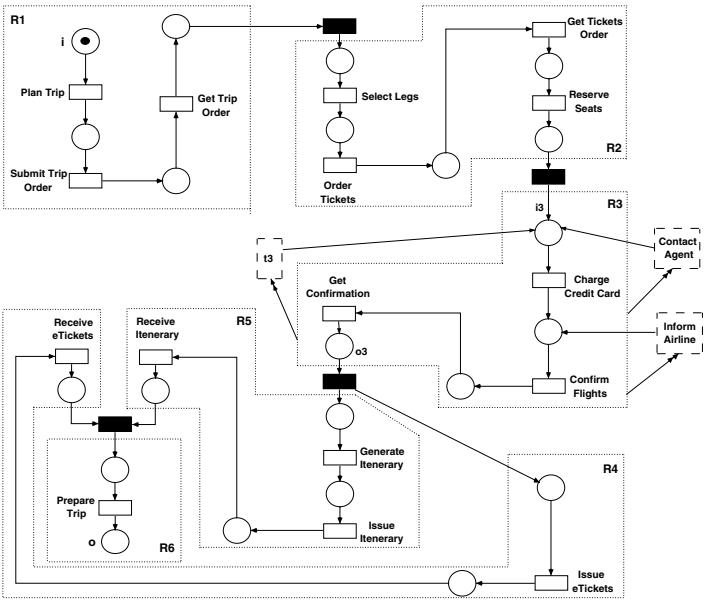

Fig. 9. Ordering Flight Tickets Workflow with Regions

This implementation has shown the validity of the approach and its advantages with regard to the traditional workflow design tools in handling and supporting hierarchical modeling. It should be noted that *HiWorD* can also be extended to interface with commonly used commercial workflow management systems such as Staffware and IBM MQSeries.

6 Conclusions

In this paper, we proposed a hierarchical model based on Petri nets to design workflows in a structured way. There are five basic refinement rules and three rules for adding communication and synchronization mechanisms. Sufficient conditions are proposed to ensure soundness after introducing any of the proposed constructs. Therefore, the workflow built according to the proposed rules is guaranteed to be a sound WF net with no risk of deadlock.

References

1. Georgakopoulos, D., Hornick, M., Sheth, A.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. Distributed and Parallel Databases **3** (1995)
2. WfMC: Workflow Management Coalition, Terminology and Glossary, Document Number WFMC-TC-1011. (1999) http://www.wfmc.org/standards/docs.htm/.

3. Chrzastowski-Wachtel, P.: Recovery Nets: Model for Dynamic Workflows. In: Proceedings of the 13th Workshop on Concurrency, Specification, and Programming, Humboldt University Report, Berlin, Germany (2002)
4. Aalst, W.v.d., Hee, K.v.: Workflow Management. Models, Methods and Systems. MIT Press, Cambridge, Massachusetts (2002)
5. Aalst, W.v.d., Hofstede, A.t., Kiepuszewski, B., Barros, A.: Workflow Patterns. Technical Report FIT-TR-2002-02, Queensland University of Technology, Brisbane, Australia (2002)
6. Aalst, W.v.d.: Verification of Workflow Nets. In Azema, P., Balbo, G., eds.: Proceedings of the Application and Theory of Petri Nets'97, Toulouse, France (1997)
7. Aalst, W.v.d.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers **8** (1998) 21–66
8. Desel, J., Esparza, J.: Free Choice Petri Nets. Volume 40 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, UK (1995)
9. Benatallah, B., Chrzastowski-Wachtel, P., Hamadi, R., O'Dell, M., Susanto, A.: HiWorD: A Petri Net-based Hierarchical Workflow Designer. In: Proceedings of the 3rd International Conference on Application of Concurrency to System Design (ACSD'03), Guimaraes, Portugal, IEEE Computer Society Press (2003)

A Case-Based Framework for Workflow Model Management

Therani Madhusudan and J. Leon Zhao

MIS Department, University of Arizona,
Tucson, Arizona 85721
madhu@email.arizona.edu
lzhao@bpa.arizona.edu

Abstract. Case-oriented workflow modeling provides flexibility in specifying and executing workflows because it is possible to consider unique, organization-specific business process conditions and thereby minimize exceptions. However, it is a laborious task for a workflow designer to derive a case-oriented workflow model from a business specification, resulting in high modeling overhead. Recent commercial systems are providing generic templates of common business processes which may be adapted to an organizations requirements. These templates, called cases, can be modified individually or multiple cases may be composed into a more complex workflow and then the assembled workflow may be modified as needed to meet the business specification. In this paper, we propose a novel framework for case-oriented retrieval, instantiation and reuse of workflow models utilizing Case-Based Reasoning (CBR) techniques. We describe key modules of a prototypical implementation to facilitate model management activities such as model retrieval, reuse and composition of component case models from a workflow repository.

Keywords: Case-oriented Workflow Modeling, Case-based Reasoning, Ad hoc Workflows, Model Reuse

1 Introduction

Business process modeling is a significant activity in enterprises as e-business and enterprise integration drive the need to deploy business processes online [7]. The business process modeling efforts are knowledge-intensive and require organizations to formalize a large number of complex inter and intra-organizational processes to facilitate their ensuing deployment in large-scale workflow systems [27]. Workflow modeling involves the translation of high-level business requirements into workflow schemas that can be executed by certain workflow engines. Specifying a workflow model is a tedious endeavor because a typical workflow model requires detailed understanding of the business process logic, the organizational chart, and the information systems accessed by the workflow. As a result, reusing existing workflow models and/or their components is highly desirable [15,14]. Recent commercial systems (such as Oracle–11*i*) are providing

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 354–369, 2003.

basic business process templates, such as order processing and supplies purchasing, that may be modified to an organizations' needs by a knowledgeable workflow designer. However, formal guidelines for reuse of these templates, rules for their instantiation in an organization or their modification and procedures for their composition into complex workflows are currently non-existent. In this paper, we present a formal framework for storage, retrieval, instantiation, reuse and composition of these generic business process templates, henceforth called cases.

The need for workflow modeling support has been recognized by researchers using approaches including a repository of organizational processes [19] and integration of organizational memory and workflow management systems [31]. However, to the best of our knowledge, there is no comprehensive approach on the retrieval and reuse of workflow models. Reuse of workflow models is especially important for ad hoc workflows where a workflow model is specified for each business case, leading to case-oriented workflow modeling [30].

Our proposed framework to facilitate reuse of workflow models is based on utilizing recent advances in Case-based reasoning techniques[16] and XML technologies. Case-based reasoning(CBR) is a problem solving technique based on the hypothesis that reasoning is reminding. That is, that problem solving utilizes past experiences. Two key tenets drive CBR approaches which are: a) similar problems have similar solutions and b) problems tend to recur in a domain. If these tenets hold in a domain, future problems are likely to be similar to current problems and current solutions may be profitably used to develop new solutions. CBR systems have proved useful in domains with weak models and large amount of unstructured, experiential knowledge. The technique provides for incremental knowledge acquisition, knowledge maintenance, increase in problem-solving efficiency (possibly with manual intervention), increase in solution quality and finally user acceptance [17,6]. Recent efforts have explored the possibility of using CBR to tailor software processes depending on the nature of the software project requirements [13]. The workflow modeling process exhibits CBR-like characteristics such as occurrence of similar business processes and reuse of similar types of data and business constraints. Experienced workflow designers derive new workflow models from existing models based on new business requirements by recognizing similar process patterns [8]. Therefore, we believe a case-oriented approach to workflow modeling is feasible. Figure 1 illustrates the CBR problem solving cycle that consists of the following steps (shown by the solid arrows): *retrieval of relevant cases*, *reuse of applicable cases*, *revision of adopted cases via testing* to ensure correctness and *retention of past solutions via learning*. At each step, knowledge is reused from the case-base (shown by the dashed by the arrow). The CBR cycle allows for both humans and automated tools to coordinate their activities in the Retrieve, Reuse, Repair and Retain phases.

A framework for instantiating the CBR reasoning cycle to enable workflow modeling is the main theme of this paper. Our framework consists of a business case ontology, a data structure for storing cases, and an algorithm for retrieving business cases and workflow schemas. The retrieval is currently performed based

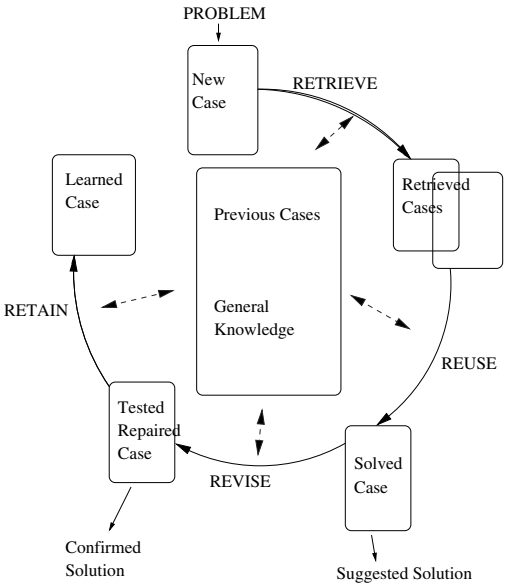

Fig. 1. The Case-based reasoning cycle

on exact matching of task characteristics. Further, once such workflow cases are retrieved, component cases are composed (assembled) into complex workflows. The composition process may be done manually or feasible assembly alternatives may be suggested in an automated manner based on AI planning techniques to the workflow designer. In our framework, we are utilizing the Hierarchical Task Network framework[23] to facilitate process composition.

The rest of the paper is organized as follows to illustrate the key aspects of our framework. In Section 2, we provide an overview of the approach with a workflow design example to illustrate the key issues and describe the overall architecture of our system. In Section 3, we describe the XML-based representation for workflow processes in our process repository and its organization. In Section 4, we present the basic approach to matching and retrieving workflow cases from a repository and enabling their ensuing composition. Approaches to verification and further adaptation are briefly outlined. Concluding remarks are in Section 5.

2 Overview of Approach

The flexibility of the CBR-approach provides a framework to facilitate the incremental definition of a workflow specification. Abstract, business level descriptions of workflow requirements can be incrementally refined into formal workflow models which may then be executed. Further, in a business context, the workflow case repository provides a means to archive and reuse best business processes, lessons learnt during their execution and variations encountered under different

business requirements. Also the case repository provides a means to store the information at multiple levels of abstraction and detail combining both structured and unstructured business knowledge. The key steps in developing a CBR-based system are:

1. Case representation development and populating the case repository with appropriate cases for a given domain.
2. Developing retrieval algorithms for case retrieval based on the notion of similarity of cues between the new problem description and the case indices.
3. Developing domain dependent procedures for facilitating reuse and adaptation of retrieved cases. It is rare that a solution can be directly applied to a new problem and usually needs to be modified before validation.
4. Developing composition procedures where cases or their pieces, also called *snippets*) may be assembled into more complex models.

We have developed a Case-Oriented Design Assistant for Workflow (CODAW) that utilizes case-based reasoning techniques by implementing each of the aforementioned steps. During the workflow specification task, a workflow designer will be provided with a high-level description of a business process (by a business process analyst) and asked to design a structured workflow. The workflow designer will use the CODAW system to retrieve past cases that embed "similar" characteristics to the new requirement. The designer analyses these cases, selects relevant cases, and composes them into a new solution. During the workflow modeling process, he may be aided by the CODAW system.

In the context of supporting workflow modeling and design, the CODAW approach provides a means to store workflow schemas and reuse the knowledge embedded in the workflow schemas. Workflow cases are of two major types: a) Prototypical-level cases which contain workflow schemas along with their design revision history, *i.e.* how the schema has evolved through time. b) Instance-level cases which contain descriptions of instances of workflow schemas that have been enacted either manually or by a system. Both types of cases may contain semistructured information that embeds design and runtime knowledge about a business process. Cases of workflow schemas embed knowledge about organizational structure, relevant roles and a variety of business constraints. Instance-level cases record information such as the actual agents involved (a particular staff member for example), tasks used, how the tasks were executed, exceptions encountered, maintenance activities, the data involved and performance metrics such as the duration of execution. In our current study, we assume that organizations follow some standards of business processes as proposed in the process handbook [19]. As a result, it is realistic to expect there is a common language in organizations on how to express the requirements in business processes with the widespread adoption of BPR techniques [12]. We further assume that the final executable workflow schemas are also specified in some given language such as web services flow language (WSFL), Business process execution language, BPEL or XML-based process definition language(XPDL) as tools for modeling these will be available in the near future [5,11,9]. Thus it is viable to envision the creation of process repositories in the near future.

The CBR-framework for Workflow Design is shown in Figure 2 wherein knowledge from process repositories may be effectively used to facilitate business process management. Numbers on the directed arrows indicate the steps of the CBR cycle described earlier, query (1), cue generation (2), trigger retrieval (3), indexing and repository access (4 & 5.1) and rank (6) and return relevant cases (7). Alternatively, the retrieved cases may be adapted and verified (8 & 5.2) or multiple cases composed (8 & 5.3) and the solution returned (9). The CBR process controller coordinates the different activities of the cycle. The figure illustrates a native XML repository to store the cases, an AI-planning module that aids composition and a verifier based on model-based checking and Petri net based techniques. The CBR process controller, the Case-base manager and the adaptation engine components of the CODAW prototype system are implemented as Lisp-based servlets embedded in JAVA components. Algorithmic details of the case retrieval and planning-based case composition are discussed in Section 4.

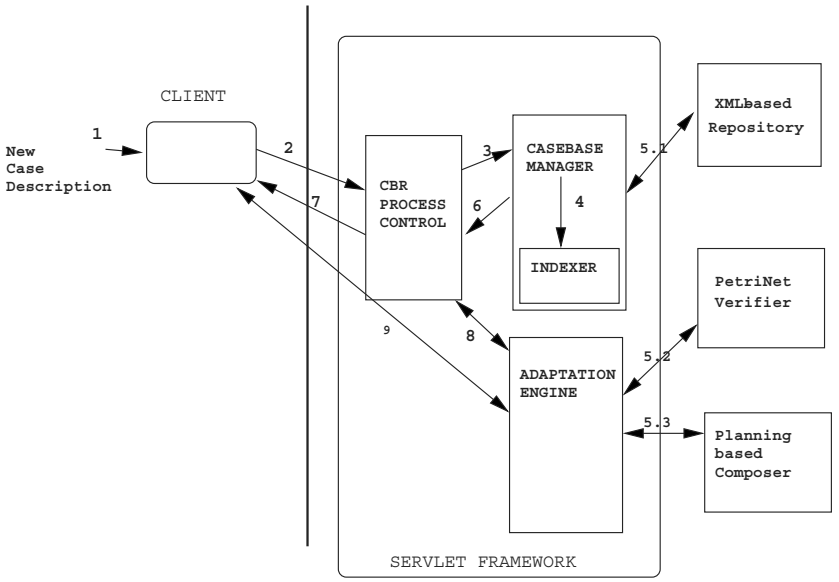

Fig. 2. Architecture of the CODAW System

We illustrate the utility of our approach with the following business process modeling scenario in the area of new product development(NPD). One of the major business processes in an organization is the product development process [21,29]. With the short product life-cycles and competitive markets in the current business environment, managing the product development process is key to organizational success. Key generic activities during product development [29] are: 1) Product planning, 2) Concept development, 3) System level design, 4) Detail

design, 5) Testing and refinement and 6) Production ramp-up. Each of these high level tasks is further refined into their subtasks in a nested manner. For example purposes, we focus on the Concept development activity. This can be further refined into the following subtasks, namely, 1) Identifying customers needs, 2) Establishing target specification, 3) Concept generation, 4) Concept selection, 5) Concept testing, 6) Setting final specifications, 7) Project planning, 8) Economic analysis, 9) Benchmarking and 10) Modeling and prototyping. The subtasks are executed in various orders ranging from the purely sequential to various concurrent schemes depending on the resources available and business constraints. It is estimated that nearly 80 percent of product costs are committed during this product development phase and managing this process effectively remains a key organizational objective to reduce overall product costs. Many organizations are moving towards standardizing these product development processes to reduce design cycle times. Coordination of the product development process is intended to be performed by a workflow system. Shown in Figure 3 is a standard development process, which follows a market driven (market-pull) approach to product development. Figure 3 illustrates an activity diagram (showing all activities and

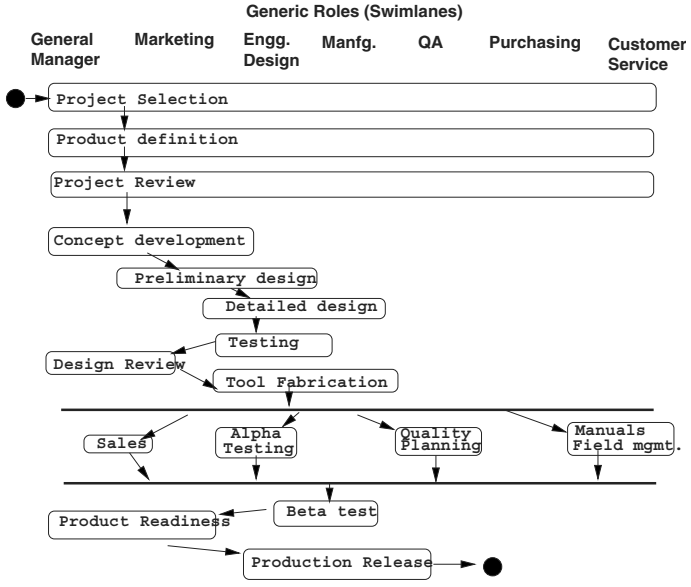

Fig. 3. A market-pull based product development process

control flow) representation of the product development process. For brevity, we have not illustrated decision blocks wherein activities may be iterative or branch. Shown at the top are the different organizational roles (swimlanes) that are involved in executing the activity. Activities may be performed by a single agent or teams of agents. The span of the rectangular boxes denote the various

roles involved in performing the activity contained with the box. For example, `Project Selection` and `Project Review` are performed by all roles. Details of each of the process steps are described in [29].

To illustrate the need for case-oriented workflow modeling, consider that the business needs evolve and changes are required to the above process. We consider a strategic change wherein internal manufacturing is being reduced and upcoming product development efforts need to consider outsourcing of manufacturing. How should the above workflow model (in Figure 3) be modified such that the tool fabrication step may be outsourced? Many alternatives may be possible. We outline a possible solution scenario below.

Shown in Figure 4 is a procurement process model within the same organization used to obtain a variety of consumable and non-consumable services and supplies for the organization. The model in Figure 4 illustrates a bidding

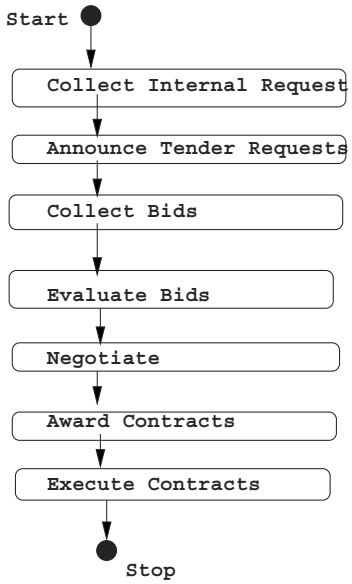

Fig. 4. A procurement process

and contracting process wherein offers from vendors, based on tender requests, are selected based on a variety of criteria. This procurement process captures the knowledge for outsourcing. To meet our new manufacturing outsourcing need wherein the tool fabrication is outsourced, a new process model can be defined as shown in Figure 5. The figure illustrates that the new process is developed by recognizing how a particular type of outsourcing via bidding may done and instantiating the same in the context of tool outsourcing. Notice that task `collection of internal requests` in the procurement process is replaced by the task `collection of design requirements` in the new process (shown in

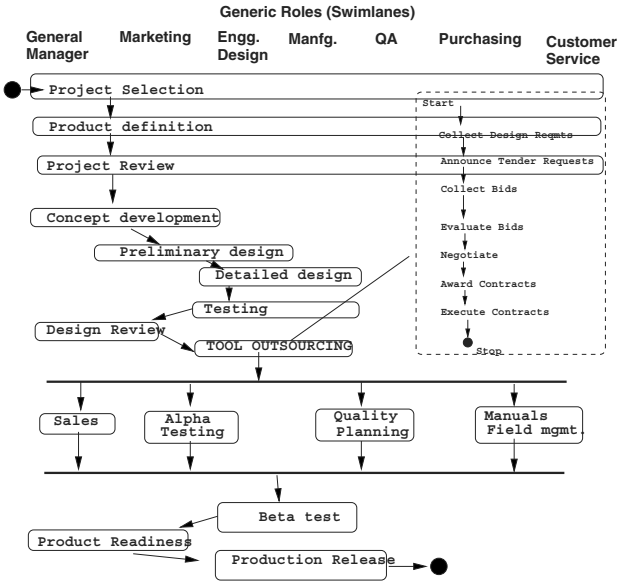

Fig. 5. A modified product development process

the dashed rectangular box in Figure 5). We believe such process generation may be possible if there is a process repository containing a model of the procurement process which may be retrieved and adapted to develop the new product development process. Another possibility is that the process repository may inherently contain a product development process model with outsourcing (say for the concept design phase) for a particular product model that may have been followed in the company. There may also be processes wherein instead of tender-based outsourcing, auction mechanisms may be followed. The process retrieval and adaptation phase provides a rich source of process modification ideas and possibilities, in contrast to a conventional approach to workflow design wherein business process reengineering is performed first and manually collected knowledge is codified. The solution obtained by retrieving process models and combining them requires additional knowledge and constraints may need to modified or added and new inconsistencies resolved.

Developing a product development workflow that handles each plausible task combination for various business scenarios by capturing all the business rules and constraints is an overwhelming task. For example, the number and type of engineers to participate on the project may vary depending on the product or business needs. Similarly, depending on the dynamic resource availability tasks may be outsourced or done inhouse. In the future world of E-business, developing such integrated processes will be a necessity and developing each process in a standalone manner would be extremely costly. However, if a process repository, storing different product development sequences as business cases, for a variety of products, manufacturing technologies, available suppliers, customer require-

ments and process costs was available, these business cases could be retrieved from the repository and a customized workflow designed for a new product development scenario. The above product development scenario illustrates the the key tenets for applicability of CBR technique, namely problem and solution recurrence.

Facilitating such process modeling and development is the aim of the CODAW system. In the following sections, we describe a workflow case representation to store process knowledge and a means to utilize this repository in the context of workflow design.

3 Case Representation, Indexing, and Repository Organization

Case representations in CBR systems take a variety of forms from OO methods to unstructured free text and multimedia [16].We have developed a semi-structured representation based on XML schemas [20]. The representation is oriented towards facilitating workflow model generation by adding additional tags and borrowing freely from standardization efforts such as XPDL, WSFL, XLANG and BPEL. Further, we have developed tools for interconversion from our representation to the above standards to facilitate enactment using XSLT [20].

The process repository in CODAW consists of two kinds of cases: a) prototypical workflow schemas as design cases and b) instances of workflow schemas as execution related cases. Shown in Figure 6 is a partial XML-based representation for the product development process. We describe the key aspects of the representation. The main tag `WorkflowSchema` consists of a three main sets of tags, a) Descriptory tags, which are, `WSID, WSName,WSType, WSDesc`, b) Workflow level Structural tags such as `TaskList, ComponentWorkflows, WorkflowInstances WFFormalModel, DesignHistory` which reflect overall process model, and c) Task level tags, which are, `TaskDesign, TaskDefn, TaskFormal`. The representation is oriented to enable declarative composition of activities and groups of activities outlined in [28]. Key ideas to note are that each workflow task is defined in three ways, namely, a) a state-space declarative AI planning based representation captured by the tag `TaskDesign`, b) A formal model representation using process algebraic representation denoted by tag `TaskFormal` , and c) A procedural task definition denoted by `TaskDefn` which highlights a procedural implementation of the task. `WFFormalModel` stores a Petri Net(PNML) based model for the overall workflow[3]. `DesignHistory` tracks all the revisions to a schema over time and justifications for the same. In the figure, we illustrate the task `Project_Selection`. This task can only be executed when a list of projects and resources are available illustrated by the predicates in `TaskDesign` and upon completion, the task has the effect of adding new projects. In terms of actual procedural code that represents this task, it is defined by `TaskDefn` including all the agents that participate in the manual execution of this task denoted by `Implementation_type`. The tag `TaskFormal` embeds a Finite State Process[18]

model of the task consisting of actions such as `init, review` and `sort` executed manually.

In a similar manner, an instance level workflow schema is also defined. Figure 7 illustrates the XML schema (in a tree-structure of the different tags) for capturing the execution history of an instance of a workflow schema. Actual data inputs and outputs, performance metrics of execution and event histories are stored. During the workflow design process, both types of process related information are used to guide the process design.

Case development and population of the case-base to bootstrap the overall system is a key task in development of the CODAW system. In general for a given domain, cases can be derived in an organization from best practices handbook, observation of execution by project teams etc. An initial set of representative cases that covers the overall population is necessary. For populating our repository, we have been collecting a variety of business cases, using the tool Togethersoft [4]. Cases are constructed in terms of UML activity diagrams and the workflow is stored in an XML format using UML XMI (XML Metadata Interchange) [24] standards from OMG. From this XML format, we extract content relevant to our tags in the representation mentioned above. Currently we have 75 design cases of various organizational business processes in our repository along with multiple execution instances for each. The persistent storage scheme in CODAW is developed based on an Open Source native XML database, namely, eXist. The storage scheme is managed by the repository manager. The repository manager handles and processes queries during the case retrieval phase as shown in Figure 2 using XQL - the XML Query Language. Cases in the repository are indexed in multiple ways. An indexing manager is an integral part of the repository. The indexing facility uses an underlying ontology for describing the cases mentioned above and an inverted indexing scheme to organize cases. Further, we also use a free text search engine such as Lucene [2] based on conventional IR techniques to index cases. Additional features of our repository are: a) Cases in the repository are organized by hierarchical taxonomies considering case-type, task types, event-types, activities, agents etc. b) Further, a domain independent ontology for describing cases and its elements is currently being developed. For example, the predicates inside the `PreConditions` tag in Figure 6 such as `(available ?x)` depend on the business ontology. c) Unstructured text is stored along with a structured workflow model such as the textual description with the tag `WFDesc`.

4 Case Retrieval and Composition

Case retrieval and composition of tasks and workflow snippets (subgraphs of a workflow) are key steps in the CODAW system. We describe the main ideas behind these two steps. Case retrieval is based on matching different aspects of the workflow and task representation to inputs provided by the workflow designer. Case composition is based on the idea that a workflow – a sequence of tasks – defines a path in a state-space implicitly defined by the preconditions

```
<?xml version="1.0"?><!DOCTYPE WorkflowSchema []>
<WSID> WS2</WSID>
<WSName> Market-Pull Workflow </WSName>
<WSType> ProductDevelopment</WSType>
<WSDesc> A product development process for a new chip </WSDesc>
<TaskList> (Project_Selection, Product_Definition,...      ) </TaskList>
<ComponentWorkflowsUnModified> WS21     </ComponentWorkflowsUnModified>
<ComponentWorkflowsModified> Null     </ComponentWorkflowsModified>
<WorkflowInstances> (WFIns1 WFIns22  WFIns23)   </WorkflowInstances>
<WFFormalModel>
<PNModel>
PN-WS2
</PNModel>
</WFFormalModel>
<Tasks>
<Task>
<TaskType> Business </TaskType>
<TaskName> Project_Selection</TaskName>
<TaskDesc>  Selects a list of new product ideas to work on </TaskDesc>
<TaskID> 1 </TaskID>
<TaskDesign>
<Parameters>
<Param> ?project_list </Param>
<Param> ?total_budget</Param>
<Param> ?resource_list</Param>
</Parameters>
<PreConditions>
<Predicate>  (available ?project_list)     </Predicate>
<Predicate>   (available ?resource_list)     </Predicate>
</PreConditions>
<PostEffects>
<Effect>  (add (new_proj_list ?new_list))        </Effect>
<Effect>   (add (new_budget ?new_budget))       </Effect>
</PostEffects>
<SubWF> WS25 - A subprocess
</SubWF>
</TaskDesign>
<TaskFormal>
<FSP>   PS = ( init -> sort_by_cost -> review -> vote -> select </FSP>
</TaskFormal>
<TaskDefn>
<Agent> General_Manager  </Agent>
<Agent> Marketing  </Agent>
<Agent> Engg_Design </Agent>
<Agent> Manfg </Agent>
<Agent> QA </Agent>
<Agent>Purchasing</Agent>
<Agent> Customer_Service</Agent>
<Procedure>
<ProcedureName> Select_Project  </ProcedureName>
<ProcedureSource> HandBook </ProcedureSource>
<Implementation_type> Manual_Team_Execution </Implementation_type>
</Procedure>
<Inputs>
<DataItem> budget       </DataItem>
<DataItem> resources       </DataItem>
<DataItem> projects        </DataItem>
</Inputs>
<Outputs>
<DataItem>  selected_projects       </DataItem>
<DataItem>  remaining_budget       </DataItem>
</Outputs>
</TaskDefn>
</Task>
```

Fig. 6. Product Development Process Case

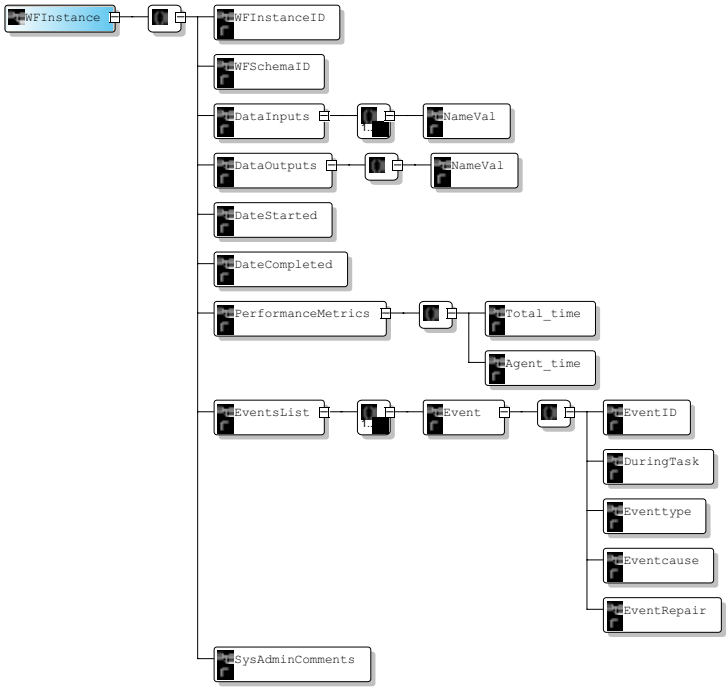

Fig. 7. Workflow Instance Case Representation

and postconditions of all the tasks in the domain. Hence, composition of tasks is equivalent to identifying tasks in a sequence that can transform an initial world state into a final required world state[16]. Two tasks can be linked in a sequence when the post-conditions of one task in a world state can trigger the ensuing task by satisfying its prececonditions. In a similar manner, two snippets of workflows can be sequenced when their post and preconditions are shared in a world state. Case-based planning is a search technique where the state space is explored starting from an initial state and tasks required to transform the initial world state into a final state are sequenced.

The case retrieval routine consists of the following steps:

– Elicit cues from new problem description: the initial query to our system can be of multiple types, unstructured or structured. Unstructured queries are a free text description of keywords from initial requirements or a semi-structured XML-based query looking for specific aspects of a case. For developing the new product development process in Figure 5, the query could be `product development contains outsourcing` wherein `contains` is a application-specific query language keyword and the rest are terms. Structured queries define preconditions or post conditions of particular task, an

initial state of a workflow or a final goal state of a workflow case or provide a graph topology of the intended workflow.
- Based on the nature of the query (structured or unstructured), multiple techniques may be used to retrieve cases. For free text queries, we use a conventional vector-space IR approach for case retrieval. For semistructured queries, since we have implemented the casebase using an XML framework, all relevant cases that match the various XML tags based on XQuery (SQL-like) framework will be retrieved. Thus using the terms `product development` and `outsourcing`, we search for subgraphs of the workflow case. The process of retrieval may return multiple number of cases which need to ranked. For structured queries, we use predicate-logic based unification of the query predicates with the predicates in the task representation of the workflows. For graph-based queries, graph isomorphism algorithms are used to retrieve the cases.
- Using a similarity metric, sort retrieved cases. The sorting of the cases can be based on a variety of similarity metrics such as a) number of common goals of workflow schemas, b) the number of common pre and post conditions between the retrieved case and the new problem, and c) a weighted sum of feature-wise similarities, where features could be domain-specific elements such as number of agents, number of inputs/outputs for a task etc.

The LISP implementation of the case-base manager in the CODAW system is based on recent algorithms on graph-based search [26,22], automated schema matching [25] and logic processing algorithms.

Failure to retrieve a similar case may trigger (manually or automatically) the case composition module in the CODAW system. Case composition is based on adapting the Hierarchical Task Network domain-independent planning framework to facilitate retrieval of specific tasks or snippets as outlined in [23]. HTN planning is a technique that creates plans by task decomposition. The planning problem is specified by an initial task network, which is a collection of tasks that need to be performed under a specified set of constraints. The planning process decomposes tasks in the initial task network into progressively smaller subtasks until the task network contains only primitive tasks or operators. The decomposition of a task into subtasks is performed using a method from a domain description. A method specifies how to decompose the task into a set of subtasks. Each method is associated with various constraints that limit the applicability of the method to certain conditions and define the relations between the subtasks of the method. HTN planning performs recursive search of the planning state space via task decomposition and constraint satisfaction. Readers are referred to [10] for further details. The CODAW system is based on the Simple Hierarchical Ordered Planner (SHOP) system [10]. SHOP has been implemented in LISP and uses a lisp-like lambda-calculus formalism. During task decomposition in CODAW, tasks and subtasks are retrieved via matching from the case-repository allowing reuse of singleton tasks and possibly workflow schemas.

Case-based composition as outlined above may not lead to a sequence of tasks that can transform an initial state to a final state. The newly composed

workflow schema may be erroneous in a variety of ways. This new schema may need to be modified to satisfy a variety of constraints relevant to the workflow. A variety of domain-independent and domain-dependent techniques are outlined in [16]. Adaptation in CODAW is equivalent to modifying the hierarchical XML structure according to various domain-independent and domain-dependent rules. In our running example, the procurement process can be adapted by changing the input data, namely, `internal requests` to `design reqmts` and propagating the changes through the procurement workflow before it is embedded in the process of Figure 5.

An adapted workflow schema may still not be completely correct and needs to be verified. For purposes of verification of a workflow case, we are exploiting standard Petri net based tools. From our XML-model, we plan to generate PNML models that can be interactively verified. Currently, we are using a tool called JARP [1] to facilitate the Petri net-based analysis. Further, inter-task interactions in a workflow can be verified. For each task, as mentioned earlier, we provide an associated formal FSP-based model to facilitate verification.

A CODAW prototype has been implemented with a focus on the XML-based repository management and SHOP-based composition. The repository has been populated with a variety of business process models obtained via business process analysis projects in real-world contexts. These initial business process models are modeled using data flow diagrams and UML activity diagrams which are then manually encoded into the XML representation. Current work is focused on developing the adaptation and verification aspects of the framework.

5 Discussion and Concluding Remarks

We have described the CODAW system, a framework for facilitating case-oriented workflow design using CBR techniques. The innovative features proposed are: a XML-based case representation for workflow models that combines free textual descriptions and a structured declarative representation, a multi-feature indexing scheme for case retrieval, a graph-based case retrieval algorithm, and an AI planning-based composition process.

The key phase in a CBR system is the retrieval step. Retrieval in a CBR system differs from conventional IR systems and databases by providing the ability to process queries on semistructured data sources. Further, database systems are designed to do exact matching and IR systems are oriented towards free text retrieval via term matching. In contrast, the goal of CBR systems is to retrieve "similar" cases. The notion of "similarity" provides the basis to embed domain related knowledge for adaptation and composiiton of cases. Furthermore, the similarity metric provides for handling approximate and incomplete queries. The quality of solutions developed by CODAW depends on the variety of workflows in the repository, the underlying business process ontology and the nature of the similarity metrics and adaptation rules. Recent research in CBR systems is focused on mixed-initiative systems wherein human-in-the-loop reasoning is performed after each step of the CBR cycle to improve the quality of solutions.

Another issue of importance is the scalability of the planning algorithm. Developing workflow design specific heuristics to speed-up planning is key to overall performance.

Currently, we are continuing to implement the prototype for recommending similar workflow models. Algorithms for verification, adaptation and composition based on manipulating the XML case representations are being tested. For future research, we assume that workflow modules (components) from similar workflow schemas can be extracted and reassembled to derive the new workflow schema. Towards this end, we will a) complete the implementation of the CODAW prototype, b) evaluate the same in real world settings and compare to current workflow design tools, and c) based on real-world evaluation, identify workflow design scenarios wherein such systems are applicable.

References

[1] JARP:Petrinets Analyzer. jarp.sourceforge.net.
[2] Lucene - a component search engine. www.apache.org.
[3] Petri Net Markup Language. www.informatik.hu-berlin.de/top/pnml.
[4] TogetherSoft: UML-based software development. www.togethersoft.com.
[5] Web services flow language. www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2002.
[6] A. Aamodt and E. Plazas. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–52, 1994.
[7] S. Aissi, P. Malu, and K. Srinivasan. E-business process modeling: the next big step. *IEEE Computer*, 35(5):55 –62, 2002.
[8] F. Casati, S. Castano, M. Fugini, I. Mirbel, and B. Pernici. Using patterns to design rules in workflows. *IEEE Transactions in Software Engineering*, 26(8), 2000.
[9] Workflow Management Coalition. XML Process Definition Language. Available at www.wfmc.org.
[10] D.S.Nau, Y.Cao, A.Lotem, and H.Munoz-Avilla. Shop:simple hierarchical ordered planner.
[11] F.Curbera, Y. Goland, J. Klein, et al. Business Process Execution Language. www.ibm.com/software/solutions/webservices/pdf/BPEL.pdf, 2002.
[12] M. Hammer. Reengineering work, don't automate, obliterate. *Harvard Business Review*, 1990.
[13] S. Henninger and K. Baumgarten. A Case-Based Approach to Tailoring Software Processes. In *Proceedings of ICCBR 2001*, number 2080 in LNAI, pages 249–262, 2001.
[14] J. Herbst and D. Karagiannis. Integrating machine learning and workflow management to support acquisition and adaptation of workflow models. In *Ninth International Workshop on Database and Expert Systems Applications*, pages 745–752, 1998.
[15] G. Joeris and O. Herzog. Managing evolving workflow specifications. In *Proceedings. 3rd IFCIS International Conference on Cooperative Information Systems*, pages 310 –319, 1998.
[16] J. L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[17] D. B. Leake, editor. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. The AAAI Press / The MIT Press, 1996.
[18] Jeff Magee and Jeff Kramer. *Concurrency*. Wiley, 1999.
[19] T.M. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C.S. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O'Donnell. Tools for inventing organizations: Towards a handbook of organizational processes. *Management Science*, 45(3):425–433, March 1999.
[20] H. Maruyama, K. Tamura, et al. *XML and JAVA: Developing Web Applications*. Addison-Wesley, 2002.
[21] A. P. Massey, M.M.Montoya-Weiss, and T.M. O'Driscoll. Performance centered design of knowledge-intensive processes. *Journal of Management Information Systems*, 18(4):37–58, 2002.
[22] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding:A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *Proceedings of the 18th International Conference on Data Engineering*, 2002.
[23] H. Munoz-Avila, D.W. Aha, D. S.Nau, R. Weber, L. Breslow, and F. Yaman. SiN:integrating case-based reasoning with task-decomposition. In *Proceedings of IJCAI*, Seattle,WA,USA, 2001. AAAI.
[24] Object Management Group. UML-XML XMI standard.
Available at www.omg.org.
[25] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema mathcing. *VLDB Journal*, 10:334–350, 2001.
[26] D. Shasha, J. T. Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Symposium on Principles of Database Systems*, pages 39–52, 2002.
[27] R. Tagg. Workflow in different styles of virtual enterprise. In *Proceedings. Workshop on Information Technology for Virtual Enterprises*, pages 21–28, 2001.
[28] G. Trajcevski, C. Baral, and J. Lobo. Formalizing and reasoning about the requirements specifications of workflow systems. *International Journal on Cooperative Information Systems*, 10(4), 2001.
[29] K.T. Ulrich and S.D. Eppinger. *Product design and development*. McGraw-Hill, NY., 1995.
[30] M. Voorhoeve and W. Van der Aalst. Ad-hoc workflow: problems and solutions. In *Proceedings., Eighth International Workshop on Database and Expert Systems Applications*, pages 36–40, 1997.
[31] C Wargitsch. Workbrain: Merging organizational memory and workflow management systems. In *Workshop on "Knowledge-Based Systems for Knowledge Management in Enterprises"*, 1997.

ADEPT Workflow Management System:*

Flexible Support for Enterprise-Wide Business Processes

– Tool Presentation –

Manfred Reichert, Stefanie Rinderle, and Peter Dadam

University of Ulm, Faculty of Computer Science,
Dept. Databases and Information Systems
{reichert, rinderle, dadam}@informatik.uni-ulm.de

Abstract. In this tool presentation we give an overview of the ADEPT workflow management system (WfMS), which is one of the few available research prototypes dealing with enterprise-wide, adaptive workflow (WF) management. ADEPT offers sophisticated modeling concepts and advanced features, like temporal constraint management, ad-hoc WF changes, WF schema evolution, synchronization of inter-workflow dependencies, and scalability. We sketch these features and describe how they have been realized within ADEPT. In addition, we show which tools and interfaces are offered to developers and users in this context. ADEPT follows a holistic approach, i.e., the described concepts have not been implemented in an isolated fashion only, but are treated in conjunction with each other by integrating them within one WfMS.

1 Introduction

Long regarded as technology for the automation of well-structured business processes, WF management is in the throes of transformation as more and more non-traditional applications require comprehensive process support. In many domains, like hospitals, engineering environments, or e-business, however, high requirements with respect to functionality, flexibility, and scalability exist [1,2, 3]. In the ADEPT project, we have addressed these requirements from the very beginning. In the meantime, we have developed an adaptive WfMS prototype, which allows users to realize flexible, enterprise-wide WF applications.

In this paper, we give an overview of the ADEPT WfMS and its related concepts, tools, and user as well as programming interfaces. Section 2 summarizes basic features of the ADEPT WfMS, which have been described in more detail in previous publications of our group [2,4,5]. In Section 3 we show how these features have been realized within the ADEPT WfMS. Section 4 sketches selected projects to demonstrate the usefulness of the developed WfMS. We conclude with a short summary in Section 5.

* This work was partially performed within the research project "Change management in adaptive workflow systems", which has been founded by the German Research Community (DFG).

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 370–379, 2003.

2 Features of the ADEPT WfMS

WF Modeling: ADEPT offers advanced concepts for the modeling, analysis, and verification of WF templates [5]. It enables the explicit definition of control and data flow, actor and resource assignments, temporal constraints, and pre-planned exceptions (e.g., forward and backward jumps [6]). This can be done in an integrated and consistent manner. Thereby ADEPT guarantees static and dynamic correctness properties (e.g., no missing input data when invoking activity programs, no undefined work assignments, no deadlocks), which is an important prerequisite for later model as well as instance changes. For control flow modeling, a simple, yet powerful formalism is offered. It is based on serial-parallel graphs with several important extensions necessary to adequately capture real-world processes. Nevertheless the resulting WF models are easy to understand for designers as well as for end users. In addition to this graph-based representation, a precise formal semantics, an equivalent operational semantics, and an efficient implementation exists.

Temporal Constraints: The handling of temporal constraints is an important feature of any WfMS. In ADEPT, designers can specify minimal and maximal durations for WF activities. At runtime, in addition, appointments may be associated with them. Furthermore, time dependencies between activities are definable (e.g., "`X` must be completed 2 days before `Y` starts"). ADEPT offers advanced concepts for specifying such constraints and for checking already at buildtime whether they are satisfiable or not [2]. Currently, we use *Temporal Constraint Networks* for representing time constraints and for checking consistency. At runtime, ADEPT schedules activities according to their starting times, supervises temporal constraints, and informs users when deadlines are going to be missed. Problems we have to deal with in this context include uncertainty, delays, and temporal inconsistencies (e.g., due to model changes).

Ad-hoc WF Changes: The support of ad-hoc changes is a must for WfMS in order to cover a broad spectrum of processes. At the instance level, ADEPT enables different kinds of ad-hoc deviations from the pre-modeled WF template (e.g., to omit activities, to change activity sequences, or to insert activities [5]). Such dynamic changes, however, must not lead to an unstable system behavior; i.e., none of the guarantees which have been achieved by formal checks at buildtime must be violated due to the dynamic change. ADEPT ensures this by introducing formal pre- and post-conditions for change operations. In particular, a consistent state must be preserved when a WF instance is going to be adapted. Additionally, ADEPT properly integrates changes with respect to authorization and documentation. Furthermore, all complexity associated with the adaptation of WF instance states, the re-mapping of input/output parameters of the components affected by a change, the problem of missing input data due to activity deletion, or the problem of deadlocks is hidden to a large degree from users.

WF Schema Evolution: In order to adequately deal with business process changes it is important that adaptations can be quickly performed at the WF type level as well. Besides versioning, ADEPT supports the propagation of WF type changes to in-progress WF instances. In doing so, change propagation

is restricted to those WF instances for which the type change does not conflict with current instance state or previous ad-hoc changes. Basic to this is a comprehensive framework for change propagation which is based on well-defined compliance criteria for WF instances and on advanced rules for automatically and efficiently adapting instance markings.

Specification and Synchronization of Inter-WF-Dependencies: Many WfMS do not provide adequate means for (semantic) inter-workflow coordination as concurrently executed WF instances are considered completely independent. Though WF templates are modeled separately from each other in order to remain comprehensible and manageable, very often corresponding instances are semantically inter-related in the one way or another [4]. Pragmatical approaches like inter-workflow message passing or merging interdependent workflows within one template do not satisfactorily solve this problem. The latter, for example, would lead to a large number of templates, each of them very complex and hard to maintain. ADEPT uses *interaction expressions* and *interaction graphs* as a simple yet powerful mechanisms for the specification and implementation of inter-WF dependencies [4]. In addition to a graph-based semi-formal interpretation, a precise formal semantics, an equivalent operational semantics, an efficient implementation, and detailed complexity analyses exist, which allow us to actually apply this formalism to coordinate inter-WF dependencies. ADEPT uses different coordination and subscription protocols to actually employ interaction expressions for the efficient synchronization of concurrent workflows.

Scalability and Distributed WF Control: In large-scale, enterprise-wide application scenarios, performance is a critical issue. Due to the high amount of communication between server(s) and clients the communication network may become a bottleneck, especially if a large amount of "long-distance" communication occurs. To avoid bottlenecks, ADEPT allows to reduce the network load by partitioning WF graphs and by migrating the control of WF instances from one server to another during run-time [7,8]; i.e., a WF instance may no longer be controlled by only one WF server. When performing such a migration, a description of the instance state is transmitted to the target server. This includes information about activity states as well as WF relevant data. To avoid unnecessary communication between servers, ADEPT allows to control parallel branches of a WF instance independently from each other (at least as no synchronization due to other reasons, e.g., a dynamic WF change, becomes necessary).

When designing these features, the following issues have been of interest: How to maintain robustness and correctness, how does the feature affect application programming, and how is it made available to the end user? In addition, we have identified the interdependencies existing between them and we have shown how the different features work in conjunction with each other.

3 ADEPT Components, Architecture, and Interfaces

We have realized the described features in the ADEPT WfMS. This research prototype supports WF control and monitoring, demonstrates the feasibility of

dynamic WF changes in a (distributed) WfMS, deals with temporal constraints, shows which user and programming interfaces are required, and proves that the concepts work in conjunction with each other as well. All system components have been implemented in Java, for communication Java RMI has been used.

3.1 ADEPT Buildtime Components

The ADEPT buildtime components enable the definition and management of WF templates, the description of inter-WF dependencies, the modeling of organizational entities, the specification of security constraints (Who is allowed to perform a particular WF change?), and the plug-in of application components. All relevant information is stored in the ADEPT repository. In addition, XML-based descriptions of model data may be generated; e.g., to export template descriptions to foreign tools or to exchange them between different WF servers. However, we do not support the XPDL syntax as defined by the Workflow Management Coalition (WfMC). On the one hand, the ADEPT WF meta model comprises several elements not captured by XPDL, on the other hand the support of WfMC standards does not have top priority in our research project.

For the modeling and management of WF templates, ADEPT offers a syntax-driven, graphical *WF editor*. A sample screen is depicted in Fig. 1. Its upper part shows a control flow window whereas the lower part displays input parameters of a selected activity and their mapping to WF data elements (data flow). Activity attributes are displayed in the right window. To each activity node a (reusable) template can be assigned. It sets out default properties like minimal/maximum duration, actor assignments (e.g., based on user roles), associated application components, and user-defined attributes. The WF designer is supported in correctly modeling and changing WF templates, i.e., static and dynamic WF properties as mentioned in Section 2 are guaranteed. To achieve this, the WF editor enables on-the-fly checks during WF editing as well as complete model checks initiated by the designer. In any case, a new WF template may only be released if all checks are successful. This is crucial for the WfMS to achieve a reliable and stable execution behavior. It is also a prerequisite for dynamic WF changes. Finally, new releases of a WF template are introduced by deploying the template to all relevant WF servers. For this, an XML-based description is sent to them and imported into their run-time databases.

ADEPT$_{distribution}$, the distributed variant of the ADEPT WfMS, additionally provides support for assigning WF servers to WF activities. This WF graph partitioning can be done manually or automatically by the use of a configuration tool. In the latter case, we make use of repository information (e.g., roles and locations of users) in order to determine optimal server assignments (i.e., to a find a partitioning which minimizes overall communication costs at run-time). Taking our example from Fig. 1, WF instances will be controlled by WF servers s_1 and s_2. (Server assignments are displayed below the activity nodes. Accordingly, "perform examination" and "write report" are controlled by s_2, whereas all other activities are carried out by s_1.)

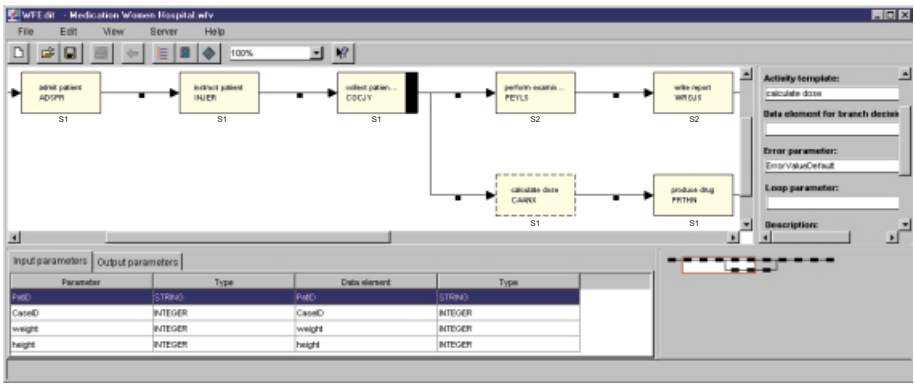

Fig. 1. ADEPT Workflow Editor

ADEPT provides several other buildtime components for defining different aspects of process-oriented information systems:

- **ADEPT interaction editor**: Powerful tool for defining and managing inter-workflow dependencies based on interaction expressions and graphs [4].
- **ADEPT organization modeler**: Graphical tool for describing organizational entities (e.g., user roles, capabilities, and organizational units) and their relationships (incl. substitution rules).
- **ADEPT application configuration tool**: This tool allows the WF designer to assign different application components to the same acitivity template. In doing so, the concrete binding of a component at runtime can be based on user as well as on workstation profiles.

3.2 ADEPT Runtime Clients

ADEPT comprises standard runtime clients for end users as well as for system and process administrators. These clients enable worklist display and manipulation, WF monitoring, activity program execution, dynamic WF changes, and system configuration.

For *worklist handling*, several client programs are available. Besides "thick" clients, ADEPT offers a Web client interface whose implementation is based on servlets. Web clients have a limited functionality when compared to standard WF clients, in particular concerning activity implementation. Both, thick and thin clients, however, already provide user interfaces for *dynamic changes*, giving end users the possibility, at run-time, to deviate from the pre-modeled task sequence. In detail, authorized actors may intervene into WF control by inserting, deleting, or shifting activities. In doing so, respective clients provide the necessary change context and allow change definition at a high semantic level. In particular, end users are not burdened with the complexity of dynamic changes; i.e., they must not deal with the problem of missing input data, the avoidance of deadlocks, or the graph transformations and state adaptations necessary to realize the change.

To monitor in-progress WF instances and to demonstrate the effects of dynamic changes, ADEPT offers a special *monitoring client*. It allows authorized users to visualize WF instance graphs, together with the information related to them. Fig. 2 shows a sample screen of a WF instance created from the template as depicted in Fig. 1. Activities "admit patient", "instruct patient", and "collect patient data" have been completed (indicated by symbol $\surd$), whereas activity "calculate dose" is currently activated (indicated by symbol $\square$). Fig. 2 also displays data elements read and written by the selected activity ("calculate dose" in the example) as well as detailed information about this activity (e.g., actor and server assignments, starting time, priority, etc.). All relevant information is managed by the WF server which controls this activity (s_1 in the example). Actually, the monitoring client only shows the WF instance graph from the viewpoint of server s_1 (to which it is connected). Normally, this server does not know how far the execution in the upper branch of the parallel branching (currently controlled by s_2) has proceeded.

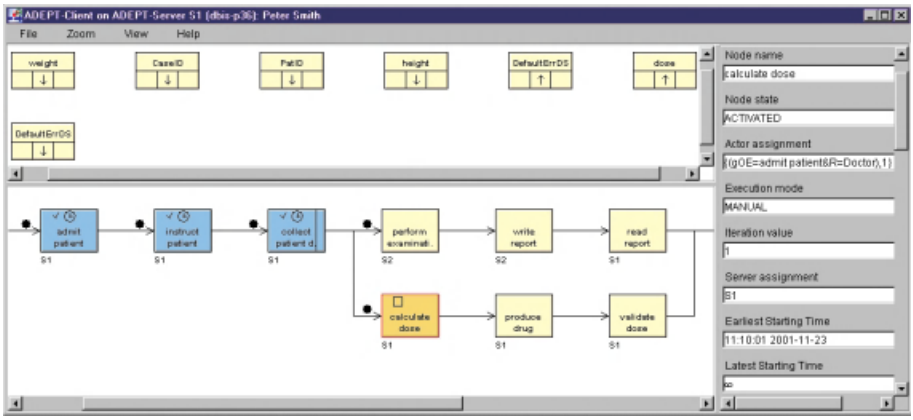

Fig. 2. ADEPT Monitoring Client (before the dynamic change of a WF instance)

Let us sketch how a dynamic change of the (distributed) WF instance from Fig. 2 is realized:

Example: Assume that an authorized user (connected to s_1) specifies that activity "perform allergy test" is to be inserted between node sets {"instruct patient"} and {"write report", "produce drug"}; i.e., the allergy test shall be performed after patient instruction and before reporting and drug production. The resulting WF instance graph is depicted in Fig. 3. Internally, the change is accomplished as follows: First of all, to decide whether the insertion is permissible or not, s_1 retrieves information about the global state of the WF instance from other active servers (s_2 in our example). As a result, s_1 finds out that activities "write report" (controlled by s_2) and "produce drug" (controlled by s_1 itself) have not been started yet; thus the dynamic insertion is allowed. In the following, s_1 performs all necessary graph transformations to realize the change. It inserts

activity "perform allergy test" parallel to the minimal block, which contains the nodes "instruct patient", "write report", and "produce drug". (For this, the AND split n_1, which represents a null task, is inserted). To enforce the desired control dependencies, three synchronization edges are added (e.g., the edge linking "perform allergy test" with "write report"). Finally, the state of the newly inserted activity is evaluated, leading to its immediate activation.

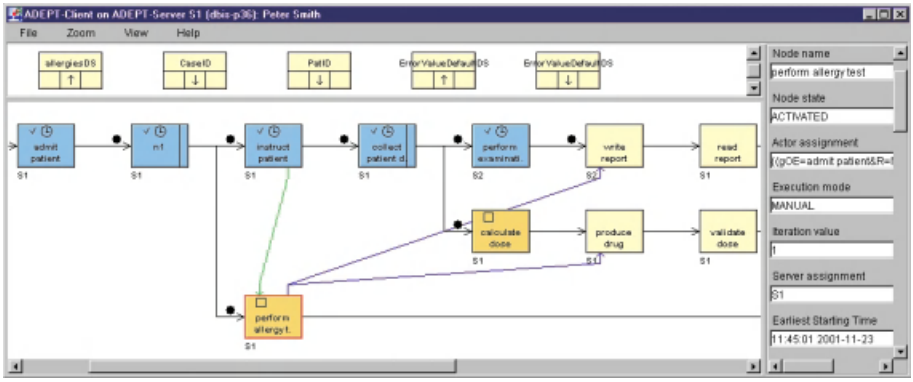

Fig. 3. ADEPT monitoring client (after the WF instance change)

3.3 ADEPT System Architecture and Programming Interfaces

The ADEPT WfMS is based on a multi-server architecture (cf. Fig. 4). A WF instance may either be controlled by a single server or by multiple servers if favorable. To each server different clients can be connected, e.g., worklist programs, monitoring components, and modeling tools. For implementing non-standard clients, ADEPT offers a rich API. It extends the one-directional client-server communication in order to enable WF servers to play an active role if need be; e.g., to initiate requests at the client site in order to get approvals from WF participants when performing a change or to immediately notify users when deadlines are going to be missed. Which communication model is used depends on the application scenario and can be configured by developers. Inter-WF dependencies are controlled by an interaction manager, which uses suitable coordination protocols to ensure that a client does not execute an action which is currently not permitted according to some inter-workflow dependency.

Server implementation is based on relational DBMS, which enables transactional execution of requests and, therefore, guarantees persistency and consistency of model and instance data. The kernel of the WF server is realized as a multi-layered architecture. The top level, the *Execution Layer*, processes client API calls (e.g., to start an activity or to perform a change). Each call is decomposed into a set of service requests from the underlying *Service Layer*, which comprises services designed along the described features (e.g., for scheduling WF activities, dynamically changing WF instances, managing user worklists,

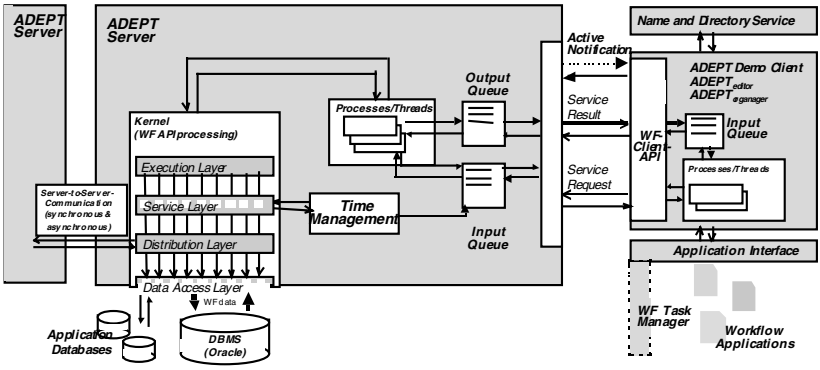

Fig. 4. ADEPT System Architecture

or handling temporal constraints). As an example take an activity completion, which leads to an update of the time schedule and the state of the respective WF instance, a role resolution of subsequent steps, and an update of worklists. Each component of the *Service Layer* itself decomposes calls into basic operations for the *Data Access Layer* (e.g., to read, to create, or to modify WF objects). Finally, if a migration of the WF control or a synchronization of the WF data becomes necessary, the *Distribution Layer* provides the required functionality.

ADEPT provides rich programming interfaces whose functionality goes far beyond the WfMC API. The offered change operations hide as much of the complexity of a dynamic change from application programmers as possible. Regarding activity insertion, for example, the method `dynamicInsertBetweenNodes` can be used: For a given WF instance, a new activity (with id `actIdentifier` and activity template `actTemplate`) can be inserted between node sets `predNodes` and `succNodes`. Information on how to map activity parameters to process data elements can be passed by the parameter `maInfo`. ADEPT allows different settings, e.g., automatic mapping of parameters to existing data elements or provision of input parameters by automatically generated, electronic forms.

```
public class WFProcessInstance {
      public WFModificationResult dynamicInsertBetweenNodes(
          ActivityTemplate actTemplate, ActivityId actIdentifier,
          InsertionArea predNodes, InsertionArea succNodes, ModificationAdjustInfo maInfo)
      // other methods
}
```

4 Practical Use and Lessons Learned

To gain concrete implementation and usability experience we have elaborated ADEPT within several research projects. Some of them have been carried out by our department in close cooperation with partners from different application

domains. Additionally, we deployed the ADEPT WfMS to other research groups who have used it as platform for implementing sophisticated WF scenarios. In summary, all these projects helped us to identify basic needs for adaptive workflows and to evolve the ADEPT WfMS over time. Current projects working with ADEPT include CONSENSUS [1], AgentWork [3], and WebFlow [9].

Clinical workflows: We consider hospital processes as being one of the most challenging application areas for WF technology [2]. Typically, a hospital comprises decentralized units which participate in a variety of medical and organizational procedures with different complexity and duration (up to several months). In a two-years WF project with a Women's Hospital, we performed an in-depth analysis of all characteristic WF types, the organizational structures and responsibilities related to them, the kinds of exceptions which may occur, and the adequate reactions necessary to deal with them. The identified requirements helped us to design the basic features of the ADEPT WfMS and to get an impression of the change facilities needed. In order to gain concrete experience with the use of WF technology in general and with the ADEPT WfMS in particular we implemented selected clinical processes based on them. The goal was to learn how computer-based process support can be smoothly integrated in the daily routine work and how adequate user interfaces have to look like. As a result, it became obvious that WfMS with a proper, secure, and robust handling of exceptional cases are a mandatory prerequisite for any WF-based clinical application. ADEPT has been perfectly coherent with these requirements.

Automatic WF adaptations: AgentWork [3] offers a system for automatically adapting WF instances. For this, a rule-based approach has been applied. When exceptional events occur, AgentWork identifies WF instances to be adapted, determines the change operations to be applied, automatically performs the change, and notifies WF participants accordingly. AgentWork has adopted the ADEPT meta model and has used the ADEPT WfMS as implementation platform. It benefits from the offered features, in particular concerning WF modeling and execution, ad-hoc changes, and temporal constraint management. Apart from this, we had received important feedback which helped us to evolve the ADEPT user and programming interfaces. Currently, with WebFlow another project of this group is on its way [9]. It aims at the flexible support of cross-organizational workflows. Due to its dynamic change facilities, the ADEPT WfMS will be a core component of WebFlow as well.

Flexible E-negotiations: CONSENSUS offers a flexible support system for e-negotiations based on parameters like quality, delivery, or warranty [1]. E-negotiations are required, for example, in conjunction with supply chains and e-procurement. On the one hand they have to be organized in a process-oriented manner, on the other hand they require flexibility and dynamism to accommodate to the various contingencies and obstacles that can appear during negotiation. For example, if a supplier or a shipping company makes a new offer that might be of interest for a buying company, the buyer will review negotiation activities already planned within the WF model and may want to rearrange them (e.g., to dynamically skip, replace, or shift activities). In this context, the ADEPT change and verification facilities have proven as perfectly coherent with

the flexibility requirements in e-negotiations. However, there are several requirements identified within the CONSENSUS project (e.g., dynamic change of WF attributes) which have not yet been fully supported by ADEPT (see [1]).

5 Summary

The ADEPT WfMS is the technological answer to the requirements set out by real-world processes. We have implemented fundamental concepts related to WF modeling, dynamic changes, temporal constraints, inter-WF dependencies, and scalability in a powerful research prototype. Currently, the integration of change propagation facilities in connection with WF schema evolution is on its way. The lessons learned from the sketched application projects have helped us to further develop the underlying concepts of the ADEPT WfMS, to improve and complement its buildtime and runtime components, and to refine user as well as programming interfaces. Adaptive WF technology as offered by ADEPT will be core of future WfMS and significantly influence the development of process-centered applications. It will drastically simplify application programming by providing rich, high-level interfaces for defining and changing model as well as instance data. As a consequence, development and adaptation times can be reduced by factors when compared to current "hard-wired" solutions.

References

1. Bassil, S., Benyoucef, M., Keller, R., Kropf, P.: Addressing dynamism in e-negotiations by workflow management systems. In: Proc. DEXA Workshop. (2002)
2. Dadam, P., Reichert, M., Kuhn, K.: Clinical workflows – the killer application for process-oriented information systems? In: Proc. 4th Int'l Conf. on Business Information Systems (BIS '00), Poznan, Poland (2000) 36–59
3. Müller, R., Rahm, E.: Dealing with logical failures for collaborating workflows. In: Proc. Int'l 5th Conf. on Coop. Inf. Sys., Eilat (2000) 210–223
4. Heinlein, C.: Workflow and process synchronization with interaction expressions and graphs. In: Proc. Int'l Conf. Data Eng., Heidelberg (2001) 243–252
5. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. JIIS **10** (1998) 93–129
6. Reichert, M., Dadam, P., Bauer, T.: Dealing with backward and forward jumps in workflow management systems. Int'l Journal Software and Systems Modeling **2** (2003)
7. Bauer, T., Dadam, P.: Efficient distributed workflow management based on variable server assignments. In: Proc. CAiSE '00, Stockholm (2000) 94–109
8. Bauer, T., Reichert, M., Dadam, P.: Intra-subnet load balancing in distributed workflow management systems. Int'l Journal of Cooperative Information Systems (accepted for publication)
9. Greiner, U., Rahm, E.: WebFlow: A system for the flexible execution of web-based, cooperative workflows (in German). In: Proc. Database Systems For Business, Technology and Web (BTW'2003), Leipzig (2003)

Modelling and Validation with VipTool

Jörg Desel, Gabriel Juhás, Robert Lorenz, and Christian Neumair*

Lehrstuhl für Angewandte Informatik
Katholische Universität Eichstätt, 85071 Eichstätt, Germany
{joerg.desel,gabriel.juhas,robert.lorenz,
christian.neumair}@ku-eichstaett.de

Abstract. This paper describes concepts and features of a new version of the VipTool. As for the original VipTool, the main issue of this software package is to generate, analyze and visualize process nets, representing the partial order behavior of business process models given by Petri nets. Whereas the original VipTool was implemented in the scripting language Python, the new VipTool is a completely new and modular implementation in Java that allows to add arbitrary extensions in a more flexible way. In this new version, several drawbacks that had appeared previously where eliminated. Moreover, the new VipTool contains additional features such as a more comfortable editor as well as eps- and XML-interfaces. The main improvement is a better support of step-wise validation of models and specifications and, alternatingly, partial verification (testing) of specification implementations. This paper also presents a small case study explaining how the VipTool supports these design steps.

1 Introduction

VipTool was originally developed at the University of Karlsruhe within the research project VIP[1] as a tool for modelling, simulation, validation and verification of business processes using Petri nets. There exist many software packages, developed at universities or software companies, which support modelling of business processes using different modelling formalisms (e. g. EPKs [8], different variants of Petri nets [1,2] etc.). Most of them, designed to analyze a Petri net model, compute the state space or use linear algebraic methods (e.g. Design/CPN, INA, Renew). The tool Woflan enables to model business processes by workflow nets [2], and to check whether the designed model fulfills the desired properties, such as soundness etc. In comparison, VipTool generates concurrent runs of a given Petri net model of a business process. If a Petri net is not too large and has only finite runs, all runs are generated. There exist some other software packages (e.g. PEP) which use the same theoretical approach of a finite and complete prefix of the unfolding of a Petri net model ([6]). But in the case the Petri net is too complex to generate the complete prefix, VipTool still

* supported by DFG: Project "SPECIMEN"

[1] Verification of Information Systems by evaluation of Partially ordered runs

W.M.P. van der Aalst et al. (Eds.): BPM 2003, LNCS 2678, pp. 380–389, 2003.

generates a substantial set of runs, because these runs are computed on the fly. Moreover, VipTool is the first tool which is able to visualize these concurrent runs.

The first version of the VipTool was written in Python [7]. The version of the VipTool presented here is a complete redesign using standard object oriented design. It is re-implemented in Java. The paper is organized as follows: In Section 2 the business process design steps supported by VipTool are described. Section 3 presents a brief description of the VipTool features. A simple case study illustrates the functionality of VipTool in Section 4. Finally, the conclusion outlines the future development.

2 The Business Process Design

VipTool was originally designed as a simulation tool for business process models. Whereas usually simulation creates and visualizes sequences of transition occurrences representing events of the business process, VipTool is based on partially ordered runs, given by process Petri nets. This concept is explained in detail in [3]. The main advantages of this approach compared to sequential simulation are shortly:

- a more efficient representation of the behavior of business processes
  (a single process net represents a set of sequences of transition occurrences which can be quite large in the presence of concurrency),
- a higher degree of expressiveness
  (the flow of objects and information is explicitly given in process nets by paths while in general it is not even implicitly given by sequential runs), and
- more efficient analysis methods for runs
  (relevant properties can be checked by efficient algorithms, exploiting the graphical structure of process nets).

One of the main issues of modelling a business process is its analysis with respect to intended properties. This analysis requires a formalization of both, the business process and the properties to be analyzed. In turn, the value of the analysis depends on the correctness of the model with respect to the actual business process and also on the correctness of the formal representation of the specification with respect to the actual intended property. To avoid confusion with the formal interpretation of the term *correctness* (a model is correct if it satisfies a property formulated by a specification), we call a model *valid* if it faithfully represents the business process. Similarly, a *valid* specification faithfully represents a relevant property. Most approaches to business process design and according tools assume validity of models and specifications and concentrate on analysis and verification issues. However, experience shows that in many cases negative results of analysis or verification are caused by invalid models or invalid specifications rather than by incorrect business processes. Even worse, positive results do not mean much if validity of models and specifications cannot be guaranteed.

The paper [4] discusses how validation of Petri net models in general can be supported by formal means and by Petri net tools. In particular, it is suggested

- to validate models by generation, visualization and inspection of process nets that represent the behavior of each system component by a process net component,
- to formalize specifications graphically within the model representation to avoid error prone new syntactical means,
- to validate specifications of a valid model by presenting separately process nets that satisfy the specification and those that do not satisfy a specification (clearly, this only makes sense if specifications can be interpreted on runs such as linear time temporal logic specifications).

Using this approach, a business process model is designed from constructive specifications, given by a Petri net that represents an early version of a business process or the environment of the business process to be developed, and declarative specifications, representing required properties of the process.

For complex business processes we suggest a step-wise procedure in [4]. The first step is creating an initial model representing the constructed specification explained above. Sometimes this model can be derived by a folding operation from known scenarios that have to be supported by the business process. In any case, the model has to be validated, as mentioned above. Then, iteratively, the following steps are performed:

- A requirement to be implemented is identified and formalized in terms of the graphical language of the model.
- This formal specification is validated by distinction of those process nets that satisfy the specification from all other process nets. This way, the question *what behavior is excluded by the specification?* gets a clear and intuitive answer. The specification is changed until it precisely matches the intended property.
- The valid specification is implemented, i.e., new elements are added to the model such that the extended model matches all previous and the new specification. Obviously, this step requires creativity and cannot be automized. However, again by generation and analysis of process nets it can be tested whether the extended model satisfies the specifications (actually, when all runs can be constructed, this test can be viewed as a verification). At this stage, other verification methods can be applied as well.
- If some requirements are still missing, we start again with the first item, until all specifications are validated and hold for the designed model.

VipTool supports all above mentioned steps. In particular, process nets representing partially ordered runs of business processes are generated from a given Petri net model of the business process. They are visualized, employing particularly adopted graph-drawing algorithms. Specifications can be expressed on the system level by graphical means. Process nets are analyzed w.r.t. these specified properties. The distinction of process nets that satisfy a specification is

supported. For the test phase, simulation stops when an error was detected. At present, the new version of VipTool only supports low-level Petri nets (whereas the previous one dealt with a restricted kind of predicate/transition nets). Also, generation of a model from a given set of runs representing known scenarios has not been implemented yet.

3 Description of the VipTool

The VipTool consists of VipEditor, which enables the user to design the business process Petri net model. The modelling is very intuitive and drawing and painting features can be used analogously as by standard Windows applications. Size, color, fonts, and other usual graphical parameters can be easily set by the user for all draw elements (places, transitions, arcs, labels, etc.). All standard editing features, such as select, move, copy, paste, undo, redo etc. are implemented in the usual way. The user-friendly environment is supported by many other features, for example by automatic alignment and by click-and-drag-points of net arcs. Usual token game simulation is also a part of the VipEditor. Figure 1 shows a screen-shot of the VipEditor with an example of a simple business process model.

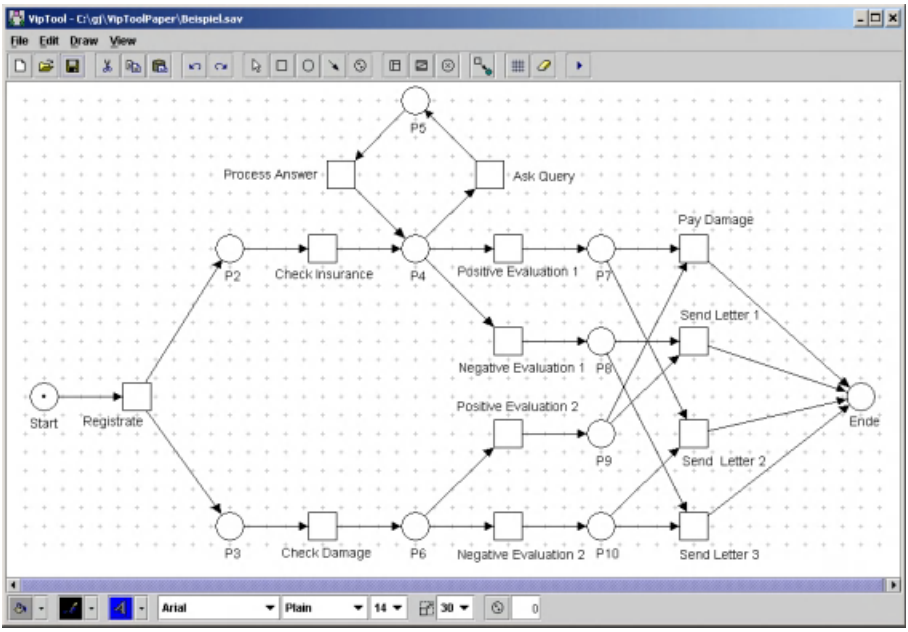

Fig. 1. Screenshot of VipEditor, including an example net, which is explained later in a case study.

An important feature of every graphical editor is the export of the image to a file in an appropriate standard graphic format. The VipTool supports exporting of pictures in Encapsulated Post Script (eps). To support the exchange of Petri nets between different tools, an XML exchange format was developed in [9]. This format is now widely accepted as a standard exchange format for Petri nets. The VipTool allows export of XML files in this format. Figure 2 shows a part of the XML file of the net from Figure 1.

Fig. 2. A part of the export of the net in Figure 1 to an XML file

The VipEngine computes the runs of the modelled Petri net. The computation of the runs of the modelled Petri net by VipEngine is based on the construction of the complete prefix of the branching process of the net. In addition to standard cut-off criteria for terminating potentially infinite runs (described in [7]), further termination criteria like bounds for the number of events or for the depth of the branching process (to be specified by the user) are implemented.

To obtain complete runs within the branching process as fast as possible, the branching process is first constructed into depth according to the following strategy: One begins with a certain starting cut (which in the beginning is the cut of conditions representing the initial marking). When trying to add a new event to the branching process, first one searches for an event enabled under the actual cut which uses conditions constructed in the previous step. If this is not possible, one searches for any event enabled under the actual cut. If there are no such events, a possible run is finished and stored, and an event is added, which is enabled under a set of already constructed concurrent conditions. Such possible events are stored in a list which is always updated after adding an event. Now one completes the next run by choosing a cut which includes the preset of that event as the new starting cut and repeating the above strategy. That new starting cut is chosen to be maximal (w.r.t. the flow relation given by the branching process) with the property that it includes the preset of the added event. In such a way, a set of runs is stored on the fly, which cover the whole branching process. Note, that in general not all possible runs are stored on the fly using this procedure, since there can be more than one possible new starting cut. Nevertheless that strategy seems to be a good trade-off between efficiency and completeness. At last the user can decide to compute the whole set of runs from the constructed branching process by applying an appropriate clique-algorithm. To guarantee, that substantial runs are computed on the fly, priority criteria can be employed in the beginning.

The runs are visualized using VipVisualizer, which is based on the Sugiyama graph-drawing algorithm accommodated in [7].

As stated above, VipTool allows to specify graphically certain properties of the business process, like specific forms of forbidden and desired behavior. Namely, the following three types of specifications are implemented (see e.g. [3]):

- Facts specify sets of forbidden markings. Facts are visualized via fact transitions.
- Goals specify two local states which have to satisfy the following property: If the first local state is reached, then the second local state will eventually also be reached. Goals are visualized using goal transitions (also known as Z-transitions, from the German word *Ziel*).
- Causal chains, which specify two transitions that are not allowed to occur causally immediately after each other. The causal chains are visualized using additional places, called common places.

4 Functionality of the VipTool: A Case Study

In this section we briefly illustrate the functionality of the VipTool by a simple case study. All figures in this section are obtained by eps-export from the VipTool.

The Petri net model of Figure 1 represents the workflow caused by a damage report in an insurance company. After the registration (transition *Registrate*) of the loss form submitted by a client, the business process divides into two

concurrent sub-processes. In the upper one, validity of the client's insurance is checked (transition *Check Insurance*), possibly further queries to the client can be answered (transitions *Ask Query* and *Process Answer*), and finally the insurance is positively or negatively evaluated (transitions *Positive Evaluation 1* and *Negative Evaluation 1*). In the lower sub-process, the damage itself is checked (transition *Check Damage*) and subsequently positively or negatively evaluated (transitions *Positive Evaluation 2* and *Negative Evaluation 2*). The two sub-processes join again. Depending on the different evaluations, the damage is payed or different sorts of refusal letters are sent to the client (transitions *Pay Damage* and *Send Letter 1-3*).

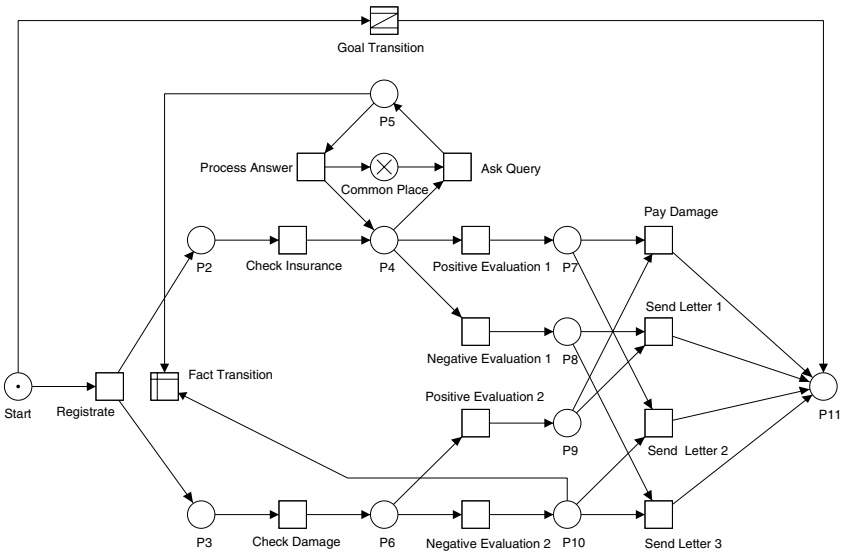

Fig. 3. The net from Figure 1 extended by graphical specifications of three different types: a fact transition, a goal transition and a common place

In Figure 3, the Petri net from Figure 1 is extended by three specifications:

- The fact transition *Fact Transition* ought to specify that we do not want to consider runs where the client is asked for further information, although the damage is negatively evaluated.
- The goal transition *Goal Transition* ought to specify that we only want to consider runs which end by the payment of the damage or the sending of one of the refusal letters. So runs which stop although further activities are possible are excluded (progress assumption) and it is assumed that eventually *Positive or Negative Evaluation* occurs (fairness assumption).

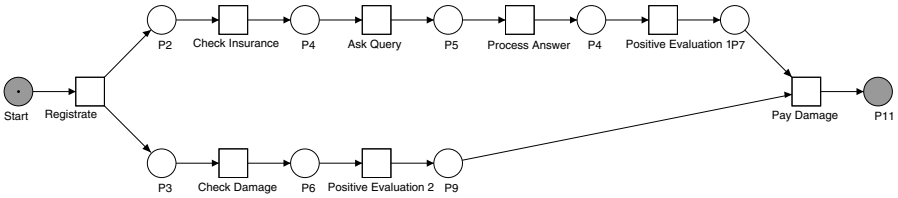

Fig. 4. A legal run of the net of Figure 3, with the places involved in the specification given by the goal transition highlighted.

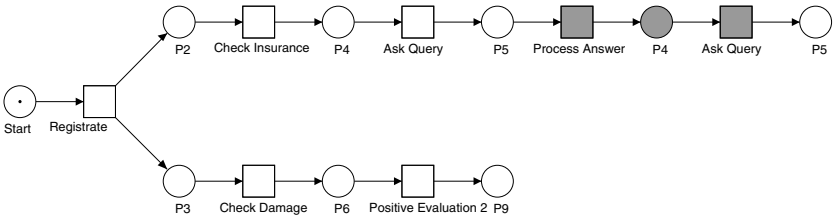

Fig. 5. An illegal run of the net in Figure 3 with the causal chain, involved by the common place, highlighted

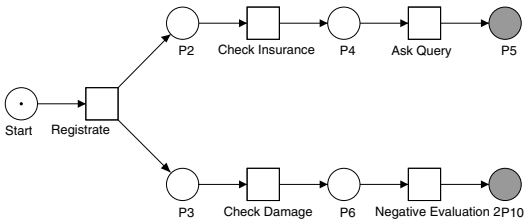

Fig. 6. A run of the net in Figure 3 which indicates that the specification given by the fact transition is wrong.

– The common place *Common Place* ought to specify that we only want to consider runs where the client is asked at most once for more information. Actually this requirement is stronger than the previous fairness assumption.

The set of runs computed by the VipEngine is divided into these runs, which fulfil the above specifications, called *legal runs*, and runs, which do not fulfil at least one of the above specifications, called *illegal runs*. Figure 4 gives an example of a legal run. The places involved by the goal transition of the above specification are highlighted by grey color. Checking the set of illegal runs, we

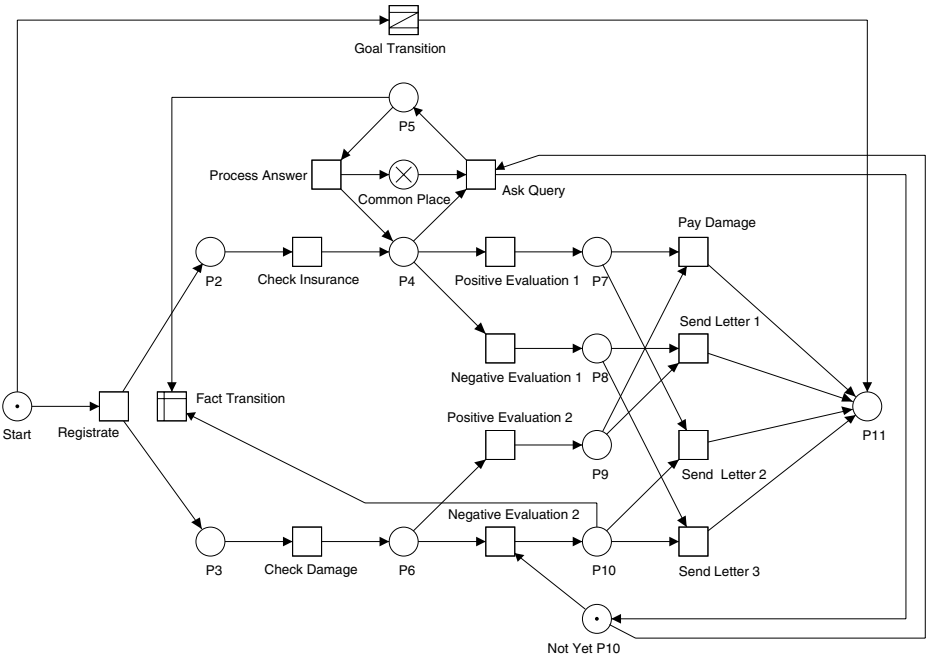

Fig. 7. Improved model of the business process. The connection of the added place *Not Yet P10* with transition *Ask Query* causes that this transition can never follow transition *Negative Evaluation 2*.

can observe whether there are desired runs which are still illegal. The Figures 5 and 6 give examples of illegal runs. In Figure 5, the computation of the run is stopped, because the specification given by the common place is violated. The causal chain (consisting of two transitions and one place) involved by the common place is highlighted by grey color.

The computation of the run in Figure 6 is aborted because the specification given by the fact transition is violated (again the involved places are highlighted). Nevertheless, we observe that a part of the behavior represented by this run should not be forbidden: Assume, the client is asked for more information (place *P5*) before the damage is negatively evaluated (place *P10*). In this case, the run should be completed by sending a refusal letter. So with this run we were able to realize that the specification given by the fact transition was badly formalized. Indeed, it is not possible to model the desired specification by one of the three implemented types of specifications. This leads to the insight that we have to change the model, to obtain the desired behavior. This is done in Figure 7. By applying again the above validation steps, one can observe now that this business process fulfils all desired requirements.

5 Conclusion

We have presented features and implementation of the new VipTool. Emphasis was on the support of step-wise design of business process models, employing validation of specifications and verification of models in each step.

The further development of VipTool includes the following tasks:

- The initial model can be obtained by folding of simpler models representing single scenarios [5]. This folding operation will be supported by the tool
- The supported business process models will be an appropriate class of high-level Petri nets (as it was the case for the original VipTool [7].
- The constructed concurrent runs will be more abstract than usual process nets, partly taking the high-level nature of tokens into account. More precisely, the concept of abstract process nets described in [4] will be implemented.
- The (experienced) user will be enabled to define further graphical specification patterns by graphically specifying classes of legal and/or illegal runs in terms of appropriate process net patterns.

Acknowledgement. We acknowledge the work of all other members of the VipTool development team: Robin Bergenthum, Thomas Liske, Thomas Loidl, Sebastian Mauser, Vesna Milijic and Niko Switek.

References

1. W.M.P. van der Aalst, J. Desel and A. Oberweis (Eds.). *Business Process Management*. Springer, LNCS 1806, 2000.
2. W.M.P. van der Aalst and K. van Hee. *Workflow Management, Models Methods and Systems.* The MIT Press, Cambridge, Massachusetts, 2002.
3. J. Desel. Validation of Process Models by Construction of Process Nets. In [1], pp. 110–128.
4. J. Desel. Model Validation - A Theoretical Issue? In J. Esparza, C. Lakos (Eds.). *Proc. of ICATPN 2002*, LNCS 2360, Springer 2002, pp. 23–43.
5. J. Desel and T. Erwin. Hybrid specifications: looking at workflows from a run-time perspective. *International Journal of Computer System Science & Engineering*, 15 Nr. 5, pp. 291–302 (2000).
6. J. Esparza, S. Römer and W. Vogler: An Improvement of McMillan's Unfolding Algorithm. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, TACAS '96*, LNCS 1055, pp. 87–106. Berlin, Heidelberg, New York: Springer (1996).
7. T. Freytag. *Softwarevaliedierung durch Auswertung von Petrinetz-Abläufen.* Dissertation, University of Karlsruhe 2001.
8. A.-W. Scheer and M. Nüttgens. ARIS Architecture and Reference Models for Business Process Management. In [1] pp. 376–390.
9. M. Weber, E. Kindler. The Petri Net Markup Language. To appear in H. Ehrig, W. Reisig, G. Rozenberg, H. Weber (Eds.). *Petri Net Technology for Communication Based Systems.* LNCS 2472, Springer 2003.

Author Index