Workshop Informatik '99

Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications

Paderborn, Germany, October 6, 1999

– Workshop Proceedings –

P. Dadam, M. Reichert (eds.)

Workshop Informatik '99

**Enterprise-wide and Cross-enterprise Workflow Management:
Concepts, Systems, Applications
Paderborn, Germany, October 6, 1999**

Program / Table of Contents

Preface

Workflow Management Systems are a relatively young technology which has the potential to change the implementation of application systems significantly. In fact, only this technology makes it possible to realize process-oriented application systems in larger quantities and at affordable costs. Very likely, in 10 to 15 years, Workflow Management Systems will be used for application development as naturally as we use database management systems for this purpose today. To reach this point, however, there is a lot to do – also at the technological basis.

The workshop aimed at bringing together researchers, developers, and applicants who deal with the application of Workflow technology for enterprise-wide or cross-enterprise applications. Focussed presentations helped to stimulate the discussion and the sharing of experiences.

The workshop was held in the context of the annual meeting of the German Informatics Society (GI) – Informatik '99 – in Paderborn.

Program Committee:

Gutavo Alonso, ETH Zürich

Peter Dadam, University of Ulm (Chairperson)

Frank Leymann, IBM Böblingen

Andreas Oberweis, University Frankfurt,

Manfred Reichert, University of Ulm

Reiner Siebert, University of Stuttgart

Gerhard Weikum, University of Saarbrücken

WISE: An Infrastructure for E-Commerce

G. Alonso   U. Fiedler   A. Lazcano   H. Schuldt   C. Schuler   N. Weiler

Swiss Federal Institute of Technology (ETH)

ETH Zentrum, Zürich CH-8092, Switzerland

E-mail: wise@ccic.ethz.ch

http://www.inf.ethz.ch/department/IS/iks/research/wise.html

May 7, 1999

1   Introduction

The Internet and the proliferation of inexpensive computing power in the form of clusters of workstations or PCs provide the basic hardware infrastructure for business to business electronic commerce in small and medium enterprises. Unfortunately, the corresponding software infrastructure is still missing. As part of the WISE project (Workflow based Interned SErvices), we have taken concrete steps to address this limitation: within WISE we have developed and deployed the software infrastructure necessary to support business to business electronic commerce in the form of virtual enterprises. The starting point was the idea to combine the tools and services of different companies as building blocks of a higher level system in which a process acts as the blueprint for control and data flow within the virtual enterprise. From here, the goal has been to build the basic support for an Internet trading community where enterprises can join their services to provide added value processes.

Following these ideas, WISE provides a working system capable of defining, enacting, and monitoring virtual enterprise business processes, as well as supporting related coordination activities. Such infrastructure includes an Internet workflow engine acting as the underlying distributed operating system controlling the execution of business processes, a process modeling tool for defining and monitoring the processes, a catalogue tool for virtual enterprise services in which to find the building blocks for the processes, and a collaborative multimedia communication environment. The project also incorporates in its design considerations about security, quality of service, execution guarantees, exception handling, high availability, and scalability, as well as diverse other aspects related to WWW based interaction, catalogue based information, catalogue search, and communication frameworks. In this regard, we have made a substantial effort to make WISE a complete solution, that is, a system incorporating all the necessary functionality to be used in practice. We firmly believe the real challenge in electronic commerce is how to provide a complete solution. In our case, this meant to develop a software tool capable of supporting the entire life cycle of a virtual business process. We see these business processes as valuable assets which need to be not only defined and enacted but also maintained, updated, and monitored. WISE supports all these chores, thereby avoiding the drawbacks of many existing products: ad-hoc and costly development, expensive maintenance, and limited applicability.
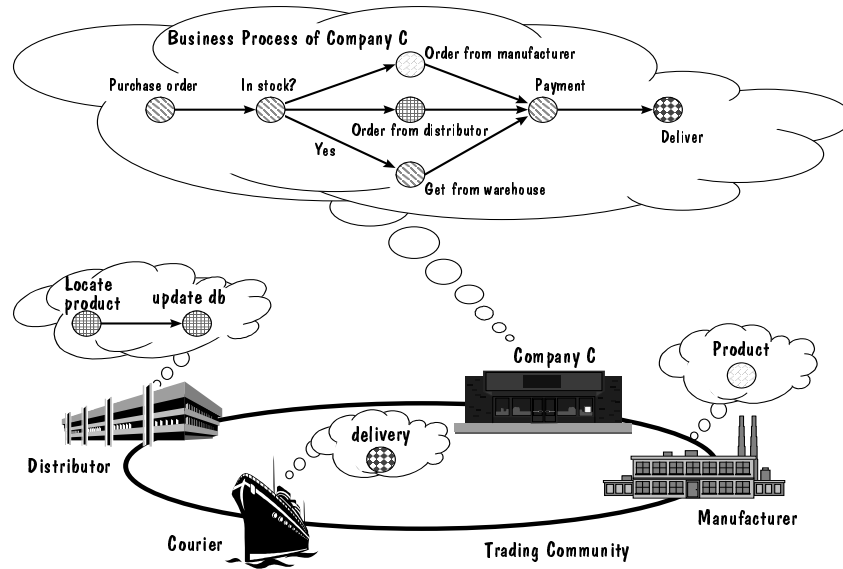
Figure 1: A company incorporating a virtual process as part of its own business processes

2 Motivation

2.1 Virtual Enterprises

The most relevant activities within a corporation are often described in the form of business processes. This is not surprising since business processes model the procedures and rules followed in order to accomplish a concrete goal (open a new bank account, obtain a credit, purchase a computer, find out the current location of a parcel, resupply shops, etc.). Following this idea, we see electronic commerce as the incorporation of information and communication systems technology into the business process to expand it beyond the corporation boundaries. In this context, we define a *virtual business process* as a business process whose definition and enactment cannot be directly tied to a single organizational entity (be it a department or a company). From here, we define *virtual enterprises* as those whose business processes are virtual business processes. Given the trend towards decentralization, we also consider that virtual business processes linking together several departments of a single organization define as well a virtual enterprise. Finally, we refer to the set of companies participating in a virtual enterprise as a *trading community*. Each member of the trading community provides a number of services to be used as building blocks for the virtual process. Based on these services, the virtual enterprise can be created by defining a virtual process in which each individual activity corresponds to one of the services provided by the participants (Figure 1).

We believe trading communities, virtual enterprises and virtual processes are a very powerful approach to interpret and identify the needs of a wide range of electronic commerce practices. For instance, in the case of retailing, a company can provide a much more sophisticated product by outsourcing aspects of the operation which are not central to its activities. A common example are companies offering a product (books, CD, flowers) without actually handling (producing, storing or delivering) the product themselves. Most of the handling is left to companies providing specialized services, which allows to significantly reduce the operational costs. The virtual enterprise model naturally captures such scenarios by simply having the distribution and delivery services incorporated as activities within the business processes of the company

2

selling the product as shown in Figure 1.

2.2 E-Comm Processes

In the example of Figure 1, independently of whether it involves mainframes and leased lines or a few PCs linked via Internet providers, an E-Comm application has many of the characteristics of a distributed computing environment. While the notions of trading community and virtual enterprise are conceptually useful, the real challenge is to use them in a software solution. Here is where the idea of process becomes relevant: the virtual business process can be seen as a distributed program running on some form of middleware linking together the resources of the trading community. These resources are the concrete applications or services offered to the virtual enterprise by the trading community and are used as the basic building blocks for the distributed program (the virtual business process). From here, the type of software to develop is the type of software that would be needed to support the definition and execution of such a coarse grained distributed program.

The analogy between an E-Comm process and a distributed program can be taken a step further. Any realistic solution to electronic commerce must take into account the true complexity of the problem. We see E-Comm processes as valuable assets needing to be properly specified, designed, developed, tested, debugged, and maintained in an effort not unlike software life-cycles. In order to do this, the language used to describe the processes must provide the necessary primitives, otherwise these tasks become extremely difficult and largely ad-hoc endeavors (as it is today).

2.3 Complete Solution

The WISE project is an integration effort with the final goal of providing a complete solution. Its architecture (Figure 2) is organized into four components (definition, enactment, monitoring, and coordination), each one of them with the role of addressing a particular issue. Thus, the *process definition* component allows virtual business processes to be defined using as building blocks the entries of a catalogue where companies within a trading community can post their services. Similarly, the *process enactment* component compiles the description of the virtual business process into a representation suitable for enactment and controls the execution of the process by invoking the corresponding services of the trading community. The *process monitoring and analysis* component is a tool keeping track of the progress made in the execution of the virtual business process and of the status of all active components in the system. The information produced by this tool is used to create an awareness model [7] used for load balancing, routing, and quality of service purposes as well as, later on, for analysis of the behavior of the process. Finally, the *coordination and communication* component supports multimedia conferencing and cooperative browsing of relevant information between all participants in the trading community using the information produced by the business process as the main source for routing.

In WISE, these four components are tightly integrated reflecting an approach to electronic commerce based on transparency and ease of use. While there is innovation in each individual component, our measure of success is the degree of integration of the system as a whole.
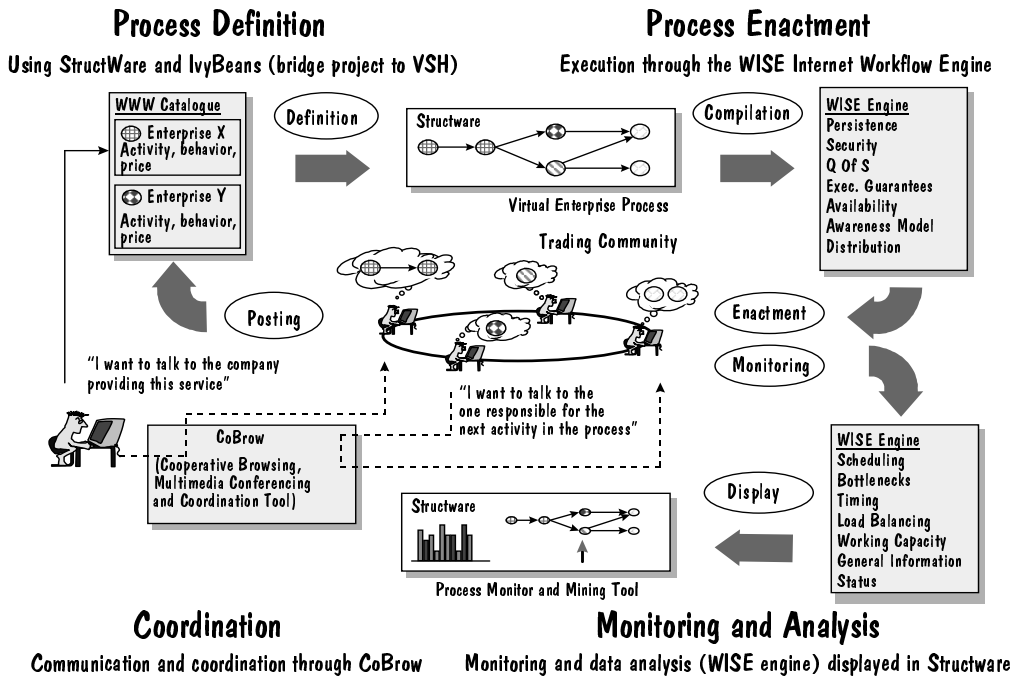
Process Definition

Using StructWare and IvyBeans (bridge project to VSH)

Process Enactment

Execution through the WISE Internet Workflow Engine

WWW Catalogue

- Enterprise X Activity, behavior, price
- Enterprise Y Activity, behavior, price

Definition

Structware

Virtual Enterprise Process

Compilation

WISE Engine
Persistence
Security
Q Of S
Exec. Guarantees
Availability
Awareness Model
Distribution

Trading Community

Posting

Enactment

Monitoring

"I want to talk to the company providing this service"

"I want to talk to the one responsible for the next activity in the process"

CoBrow
(Cooperative Browsing, Multimedia Conferencing and Coordination Tool)

Display

WISE Engine
Scheduling
Bottlenecks
Timing
Load Balancing
Working Capacity
General Information
Status

Structware

Process Monitor and Mining Tool

Coordination

Communication and coordination through CoBrow

Monitoring and Analysis

Monitoring and data analysis (WISE engine) displayed in Structware

Figure 2: The different components of the WISE project

3 Architecture and Components

3.1 Modeling

In WISE, virtual business processes are constructed by using the services offered by different companies as building blocks. The virtual business process integrates the services of the different companies establishing the order of invocation, the control logic and the data flow between the participants in the same way a workflow process orchestrates business models within a single corporation. To make this idea a reality, there are two elements that WISE must provide. The first is a mechanism for the participants to publish their services. The second is a way to define a process based on such services. For these purposes, WISE uses a WWW catalogue and a business process modeling tool (Figure 2).

The WWW catalogue uses Java applet/servlet technology to allow companies in the trading community to advertise their services and to "see" the semantics of the services provided by other companies [15]. The catalogue contains objects encapsulating the behavior of each service. A Java version of a business modeling tool supporting simulation and analysis (see below) is then used to allow a company to see the exact characteristics of each entry in the catalogue. When a company wants to make an entry in the catalogue, it specifies the service using the modeling tool.

From the catalogue, a drag and drop type of interface is used to build the virtual business process. The tool we use for process definition is *Structware* [12], a product of IvyTeam, one of the partners in the project. Structware, which is internally based on Petri-nets, supports not only the modeling of business processes but also sophisticated analysis of its behavior (bottlenecks, average execution times, costs, delays, what if analysis, etc.). In terms of process definition, Structware supports the standard flow control primitives

4

of a workflow tool. It is possible to define conditional branching, nested processes, and assign additional information to each task within the process. This last point is important from the point of view of WISE since it allows to use this additional information as the configuration information necessary to enact the process

We see this entire procedure as a form of high level, coarse grained programming. We have successfully applied this idea of "workflow programming" within WISE and other projects in order to provide sophisticated language primitives not available in commercial workflow tools. For instance, we can provide a complete exception handling capability [10], an event handling mechanism and inter-process communication [11]. This functionality is missing in current systems and we consider it to be crucial in realistic environments.

3.2 Enactment

The enactment of the virtual business processes is performed by the WISE engine, which is based on work done within the OPERA project [9, 2]. The WISE engine extends ideas from workflow management [6, 3], and uses known techniques for distributing this functionality [18, 13, 4]. In addition, a considerable amount of extensions have been introduced to make workflow a suitable foundation for electronic commerce (for a different approach to electronic commerce based on workflow technology see [16]). Among them, there are three that deserve special attention: security, quality of service, and execution guarantees.

Given the nature of the data exchanged between the different participants in the trading community, WISE incorporates the necessary security mechanisms in the form of encryption of data for transmission over the network as well as a complete set of authentication measures for both execution, access, and monitoring of the processes. Also, to make the notion of trading community viable given the current limitations of bandwidth, the WISE engine incorporates quality of service guarantees based on execution statistics and network characteristics. Our current approach is based on distinguishing different process categories (critical, important, normal) and providing for each of them a different quality of service. Finally, the WISE engine also incorporates execution guarantees, whereby a process is always guaranteed to finish in a consistent state either by removing all changes it has introduced or by forcing it to terminate following a sequence of actions with a pre-determined outcome [17]. The execution guarantees are based on the notion of spheres of atomicity and isolation [8, 5, 1, 14], which allow us to specify which parts of the business process need to be made atomic for recovery purposes and which parts of the process need to be isolated from interferences of other processes.

3.3 Audit and Monitoring

WISE provides tools to find out the status of any running process in the system in order to allow users to keep track and troubleshoot them when necessary. In addition, process design is a difficult task. In virtual enterprise environments it is difficult to foresee all possible eventualities until some example runs are available. Process design is an iterative procedure where WISE can be of great help by providing accurate measurements of all the characteristics affecting the execution of a process: overall duration, bottlenecks, relative duration of each task with respect to the duration of the entire process, loads at each participant site, deadlines missed, and so forth.

In order to provide this functionality, WISE incorporates the necessary modules within the execution

5

engine to keep track of executing processes. In addition, it uses a history space where information about all already executed processes is stored and organized in a way that facilitates its analysis. For displaying this information, we plan to take advantage of the capabilities of Structware. In the same way that a Structware process is compiled and translated into notation understandable by the WISE engine, the information produced by the WISE engine will be translated into the appropriate format to be displayed using Structware's interface.

Finally, WISE will also include an awareness model [7] that will allow the engine to make decisions based on its own status and that of the participants. This awareness model is necessary for load balancing, increased availability, conflict resolution, notification mechanisms, and the handling of exceptions.

3.4 Coordination

Unlike in conventional workflow engines, WISE will operate in an environment where the different participants and the different elements of the process are not necessarily in a position to easily exchange information among them. Note that, as the concept of trading community implies, each participant could be not only on a different location but in an entirely different company. It is nevertheless important for the participants to be able to communicate in order to resolve the unavoidable inconsistencies and minor problems associated with any process (Figure 2). An essential aspect of this communication and collaboration is that it will be context based. That is, a user will not necessarily ask to communicate with a concrete person but, rather, with the person who played a given role in the execution of the process. To achieve this goal, WISE uses the results of the CoBrow (Collaborative Browsing in Information Resources) project [19].

4   The WISE system

The current version of WISE uses IvyFrame (a commercial product of IvyTeam) as front end, both for the definition and the monitoring of processes. WISE is platform independent (Server runs on UNIX, Clients on UNIX, OS/2, and Windows) and can interact with a variety of applications (existing interfaces include SAP R/3 and IBM FlowMark). From a practical point of view, WISE can be used as a generic workflow engine but its real potential lies as an engine for electronic commerce. One possible scenario for the deployment of WISE is as the central tool for a company providing support for other companies wanting to engage in electronic commerce but not willing or able to do the necessary investments in resources and expertise. Another possibility is to use WISE as a tool for implementing value added business processes so that a company can offer new services by combining services provided by other companies. In particular, the application to virtual store fronts for generic customer services, computer equipment, bookstores, and appliances is immediate. We are currently working on supporting other possible scenarios related to payment protocols and electronic document exchanges.

5   Conclusions

In this extended abstract, we have presented a basic infrastructure for business to business electronic commerce. In this form of e-commerce, different companies join their services to form a virtual enterprise, which provides a business process that can be executed over the Internet. WISE includes different com-

ponents to define, enact and monitor visual enterprise processes, supporting also the communication and coordination between the participants.

WISE should be seen as an integration effort where several known technologies as well as new ideas are being brought together in order to provide a coherent technological solution. We expect that the results of the project will both enhance considerably the scope of application and expressive power of current workflow systems and open up significant opportunities in the area of electronic commerce.

Project Data

References

[1] G. Alonso, D. Agrawal, and A. El Abbadi. Process Synchronization in Workflow Management Systems. In *8th IEEE Symposium on Parallel and Distributed Processing (SPDS'96). New Orleans, USA.*, October 1996.

[2] G. Alonso, C. Hagen, H.J. Schek, and M. Tresch. Distributed Processing over Stand-alone Systems and Applications. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB'97)*, Athens, Greece, August 1997.

[3] F. Casati, P. Grefen, B. Pernici, G. Pozzi, and G. Sanchez. WIDE Workflow Model and Architecture. Technical Report 96-19, University of Twente, 1996.

[4] Stefano Ceri, Paul W.P.J. Grefen, and Gabriel Sanchez. WIDE: A Distributed Architecture for Workflow Management. In *Proceedings 7th International Workshop on Research Issues in Data Engineering (RIDE'97)*, pages 76–79, Birmingham, UK, April 1997.

[5] D. Georgakopoulos and M. Hornick. A Framework for Enforceable Specification of Extended Transaction Models and Transactional Workflows. *International Journal of Intelligent and Cooperative Information Systems*, 3(3), September 1994.

[6] D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, April 1995.

[7] Dimitrios Georgakopoulos. Collaboration management infrastructure for comprehensive process and service management, May 1998. Presentation in International Symposium on Advanced Database Support for Workflow Management, Enschede, The Netherlands.

[8] Dimitrios Georgakopoulos, Mark Hornick, Piotr Krychniak, and F. Manola. Specification and Management of Extended Transactions in DOMS. In *RIDE-IMS'93*, pages 253–257, Vienna, Austria, April 1993.

[9] C. Hagen. Atomarität in Workflow- und Prozessunterstützungssystemen. In *9. GI-Workshop "Grundlagen von Datenbanken"*, Friedrichsbrunn, Germany, May 1997. In German.

[10] C. Hagen and G. Alonso. Flexible exception handling in the OPERA process support system. In *Proc. of the 18th Intl. Conference on Distributed Computing Systems*, Amsterdam, The Netherlands, May 1998.

[11] C. Hagen and G. Alonso. Beyond the black box: Event-based inter-process communication in process support systems. In *Proc. of the 19th Intl. Conference on Distributed Computing Systems*, Austin, Texas, USA, May 1999.

[12] IvyTeam. Structware'98 Process Manager. Available through http://www.ivyteam.com, 1998.

[13] S. Jablonski and C. Bussler. *Workflow Management*. International Thomson Computer Press, 1996.

[14] Sushil Jajodia and Larry Kerschberg, editors. *Advanced Transaction Models and Architectures*, chapter 1, pages 3–34. Kluwer Academic Publishers, 1997.

[15] H. Lienhard. IvyBeans - Bridge to VSH and the project WISE. In *Proceedings of the Conference of the Swiss Priority Programme Information and Communication Structures, Zürich, Switzerland*, July 1998.

[16] P. Muth, J. Weissenfels, and G. Weikum. What Workflow Technology Can Do For Electronic Commerce. Technical report, University of the Saarland, Department of Computer Science, Saarbrücken, Germany.

[17] H. Schuldt, G. Alonso, and H.-J. Schek. Concurrency Control and Recovery in Transactional Process Management. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'99)*, Philadelphia, PA, May 31 - June 2 1999.

[18] H. Schuster, S. Jablonski, T. Kirsche, and C. Bussler. A Client/Server Architecture for Distributed Workflow Management Systems. In *Proc. of Third Int'l. Conf. on Parallel and Distributed Information Systems*, Austin, Texas, September 1994.

[19] G. Sidler, A. Scott, and H. Wolf. Collaborative Browsing in the World Wide Web. In *Proceedings of the 8th Joint European Networking Conference, Edinburgh, Scotland*, May 1997.

Give me all I pay for — Execution Guarantees in Electronic Commerce Payment Processes

Heiko Schuldt   Andrei Popovici   Hans-Jörg Schek

Database Research Group
Institute of Information Systems
ETH Zentrum, 8092 Zürich, Switzerland
Email: {`schuldt,popovici,schek`}`@inf.ethz.ch`

Abstract

Electronic Commerce over the Internet is one of the most rapidly growing areas in todays business. However, considering the most important phase of Electronic Commerce, the *payment*, it has to be noted that in most currently exploited approaches, support for at least one of the participants is limited. From a general point of view, a couple of requirements for correct payment interactions exist, namely different levels of atomicity in the exchange of money and goods of a single customer with different merchants. In this paper, we identify the different requirements participants demand on Electronic Commerce payments from the point of view of execution guarantees and present how payment interactions can be implemented by transactional processes. Finally, we show how these execution guarantees can be provided for payment processes in a natural way by applying the ideas of transactional process management to an Electronic Commerce *Payment Coordinator*.

1 Introduction

Along with the enormous proliferation of the Internet, Electronic Commerce (E-Commerce) is continuously gaining importance. The spectrum of applications that are subsumed under the term E-Commerce leads from rather simple orders performed by Email to the purchase of shopping baskets consisting of several goods originating from different merchants by spending electronic cash tokens.

Remarkably, E-Commerce is a very interdisciplinary research area. As existing approaches are powered by different communities (i.e., cryptography, networking, etc.), they are very heterogeneous in nature and thus always focus on different special problems. From the point of view of the database community, atomicity properties have been identified as one key requirement for payment protocols in E-Commerce [Tyg96, Tyg98]. The more complex interactions with consumers and merchants become, the more dimensions of atomicity have to be addressed. In the simplest case, only money has to be transferred atomically from the consumer to the merchant. However, considering complex shopping baskets filled with (electronic) goods from several merchants, atomicity may also be required for the purchase of all these goods originating from different possibly independent and autonomous sources, along with the atomic exchange of money and all goods.

Due to their distributed nature, protocols that have been suggested to support payment atomicity in E-Commerce impose high requirements on the participating instances (e.g., NetBill [CTS95]). However, with a centralized payment coordinator, the complex interactions of the various participants can be embedded within a payment process, thus reducing the prerequisites for merchants and customers to participate in E-Commerce. Transactional process management [SAS99] can then be exploited in order to provide the necessary execution guarantees for transactional E-Commerce payment processes in a natural way.

This paper is structured as follows: In Section 2, we provide a general framework for E-Commerce payment interactions. Based on this framework, we analyze the different atomicity requirements for E–Commerce payment (Section 3). Then, in Section 4, we summarize transactional process management and

1

present the structure of a transactional payment process allowing the required execution guarantees to be provided by a Payment Coordinator. Section 5 finally concludes the paper.

2 Schema for Payment Protocols in E-Commerce

The description of sales interactions in non-electronic markets [Sch98] encompasses three phases: information, negotiation, and payment. During the information phase, a customer evaluates and compares the offers of several merchants. After selecting the best offer, she negotiates with the chosen merchant the conditions for the deal (negotiation). If they reach an agreement, the last step (the payment) involves the money transfer from customer to merchant and the service (the merchant fulfills his contract).

Most electronic payment systems focus only on the money transfer of the last phase. Our view of an *electronic payment scheme* also considers the systems and protocols for accomplishing both the money transfer and the service.

2.1 Participants

An electronic payment scheme involves participants originating from two distinct worlds: on the Internet side there are the customer, the merchant and a third entity, the payment server which coordinates the two. The other side is represented by the financial world with its proprietary network infrastructure and protocols. The participants are financial institutes and again the payment server, that has to consistently transform the data flow on the Internet side in corresponding "real world" money flow. The participants are depicted in Figure 1.

2.2 Steps of an E-Commerce Transaction

Prior to the payment transaction, the participants are involved in an *initialization* phase, depicted in Figure 1 by dashed arrows. Both customer and merchant have to establish accounts within the financial institutes "issuer" (or "acquirer", resp.). The transformation of electronic money into real money is performed using these accounts. Also in this phase the customer receives from his bank a *customer secret* which enables him to perform electronic payments. The customer secret is visible only for the customer herself, for the issuing bank and (eventually) for the payment server. The most common form of the customer secret is a credit card number, in electronic cash schemes (such as eCash^{TM} [Dig99]), the customer secret is an E-cash token. Because account operations are rather less often than payments, we can consider them as part of the initialization phase.

Almost all the payment schemes contain the five following steps, marked in Figure 1:

- Negotiation (1): the customer selects the desired service or merchandise she wants from the merchant, and negotiates with the merchant the price of the service. The result of this step is the *Order Information*. The Order Information is a protocol of the negotiation phase, including service (merchandise) and price specification.

- Payment order (2): the customer sends Payment Information (PI) and Order Information (OI_c) to the merchant. The OI_c is the customer's view of the agreement with the merchant.

- Payment authorization (3): the merchant forwards PI, OI_c, OI_m and additional data to the payment server. OI_m is the merchant's view of the agreement with the customer.

  The payment server directly or indirectly verifies the validity of the payment information, the consistency of the payment using OI_c and OI_m. It eventually triggers the real world money transfer using its role on the non-Internet side.

  At the end of the payment authorization, the merchant receives a confirmation message C from the payment server (4).

- Purchase response (5): The merchant sends himself a confirmation to the customer. In case of electronic (non-tangible) goods, the purchase response can be immediately followed by the merchandise or the service itself.
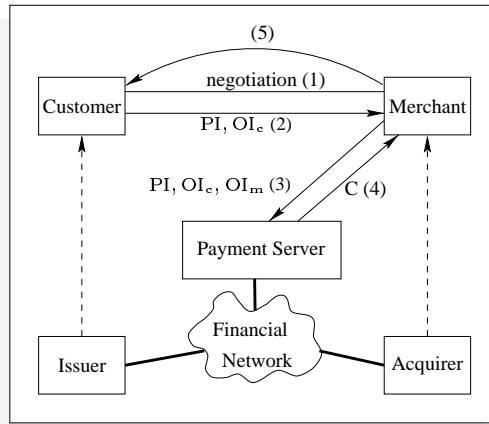
Figure 1: Generic payment steps

In most existent payment protocols, the payment server is invoked by the merchant. This is no intrinsic restriction, and communication between customer and payment server is also possible.

2.3 Characteristics of Payment Protocols

Several criteria serve as classification models of electronic payment schemes. Starting from the moment of transformation of real money into electronic money, payment protocols can be split in *pre-paid systems* and *pay-by-instruction* ones. *Atomicity* is another item, which will be discussed in detail later. Some protocols introduce the notion of *provability*, which is the ability of each party to prove their correct interactions. *Anonymity* is especially addressed by cash-based-systems. There are also implementation issues like *scalability*, *flexibility*, *efficiency*, *ease of use* and *off-line operation*, which are also important because of the large number of users expected.

3 Atomicity in Electronic Commerce

One key requirement in E-Commerce is to guarantee atomic interactions between the various participants in E-Commerce payment. As E–Commerce and thus also payment takes place in a highly distributed and heterogeneous environment, various aspects of atomicity can be identified: aside of money and goods atomicity [Tyg96, Tyg98], also the atomic interaction of a customer with multiple merchants is needed. In what follows, we analyze and classify these different atomicity requirements in detail.

Money Atomicity   The basic form of atomicity in E-Commerce is associated with the transfer of money from the customer to the merchant. This is denoted by the term *money atomicity* [Tyg96]. As no viable E-Commerce payment solution can exist without supporting this atomicity property, multiple solutions have been proposed or are already established [MV96, Dig99]. However, the atomicity property is tightly coupled with the protocol architecture and design.

Certified Atomic Delivery   Aside of money, also goods have to be transferred. Therefore, a further requirement is that the delivery takes place atomically. This can even be reinforced in that both associated parties –customer and merchant– require the necessary information in order to prove that the goods sent (or received, resp.) are the ones both parties agreed to in the initial negotiation phase (*certified atomic delivery*, encompassing the goods atomicity and the certified delivery described in [Tyg96]). This strengthened requirement results from the fact that –in contrast to traditional distributed database transactions where only technical failures have to be addressed– in E–Commerce also fraudulent behavior of participants has to be coped with. Especially when dealing with goods that can be transferred electronically, the combination of money atomicity and certified delivery is

3

an important issue. In [CHTY96], this is realized by a customized Two-Phase-Commit protocol [GR93].

Distributed Purchase Atomicity In many E-Commerce applications, interaction of customers is not limited to a single merchant. Consider, for instance, a customer who wants to purchase specialized software from a merchant. In order run this software, she also needs an operating system which is, however, only available from a different merchant. As both goods individually are of no value for the customer, she needs the guarantee to perform the purchase transaction with the two different merchants atomically in order to get both products or none. *Distributed purchase atomicity* addresses the encompassment of interactions with different independent merchants into one single transaction.

Most currently deployed payment coordinators support only money atomicity while some advanced systems address also distributed purchase atomicity. However, all three dimensions are –to our best knowledge– not provided by existing systems and protocols although the highest level of guarantees would be supported and although this is required by a set of real-world applications.

This lack of support for full atomicity in E-Commerce payment is addressed by our current research activities where we apply transactional process management (section 4) to realize an E-Commerce Payment Coordinator.

4 Transactional Processes for E-Commerce Payments

In this section, we introduce the theory of transactional process management that provides a criterion for the correct execution of processes with respect to recovery (when failures of single processes have to be considered) and concurrency control (when multiple parallel processes access shared resources simultaneously) and we point out how this theory can be applied for payments in E–Commerce.

4.1 Transactional Process Management

In conventional databases, concurrency control and recovery are well understood problems. Unfortunately, this is not the case when transactions are grouped into entities with higher level semantics, such as *transactional processes*. Although concurrent processes may access shared resources simultaneously, consistency has to be guaranteed for these executions.

Transactional process management [SAS99] has to enforce consistency for concurrent executions and, at the same time, to cope with the added structure found in processes. In particular, and unlike in traditional transactions, processes introduce flow of control as one of the basic semantic elements. Thus, it has to be taken into consideration that processes already impose ordering constraints among their different operations and among their alternative executions. Similarly, processes integrate invocations to applications with different atomicity properties (e.g., activities may or may not be semantically compensatable).

The main components of transactional process management consist of a coordinator acting as top level scheduler and several transactional coordination agents [SSA99] —one for each subsystem participating in transactional processes— acting as lower level schedulers. Processes encompass *activities* which are invocations in subsystems scheduled by the coordinator. The coordinator's task is to execute transactional processes correctly with respect to concurrency control and recovery. Firstly, the execution guarantees to be provided include guaranteed termination, a more general notion of atomicity than the standard all or nothing semantics which is realized by partial compensation and alternative executions. Secondly, the correct parallelization of concurrent processes is required and thirdly, by applying the ideas of the composite systems theory [ABFS97], a high degree of parallelism for concurrent processes is to be provided.

The key aspects of transactional process management can be briefly summarized as follows: The coordinator acts as a kind of transaction scheduler that is more general than a traditional database scheduler in that it i.) knows about properties of activities (compensatable, retriable, or pivot, taken from the flex transaction model [MRSK93, ZNBB94]), ii.) knows about alternative executions paths in case of failures, and iii.) knows about semantic commutativity of activities.

Based on this information, the coordinator ensures global correctness but only under the assumption that the activities within the processes to be scheduled themselves provide transactional functionality (such as, for instance, atomicity, compensatability, order-preservation, etc.).

4.2 Transactional Payment Processes

According to [MWW98], trade interactions between customers and merchants can be classified in three phases: pre-sales, sales and post-sales. While the sales phase has a well-defined structure (especially the payment processing, see section 2), this is in general not the case for the pre-sales and the post-sales phase. Due to this well-defined structure, processes are a highly appropriate means to implement the interactions that have to be performed for payment purposes. Furthermore, all atomicity requirements for payments in E–Commerce can be realized in an elegant way by applying the ideas of transactional process management in an *E-Commerce Payment Coordinator*.

These processes are extensions of anonymous atomic transactions described in [CHTY96], they rely on electronic cash token as means of payment, and are primarily designed for the purchase of electronically available goods that are transferred in an encrypted way to the customer prior to the payment. Furthermore, the idea of transactional payment processes is to encompass all interactions between the participants (customer, merchants and bank). To this end, and in contrast to the currently applied payment schemes, the payment has to be initiated by the customer by invoking a payment process at the Payment Coordinator[1]. The structure of a transactional payment process can be seen in figure 2. The precedence orders are depicted by solid arcs while for the preference order, dotted arcs are used. For each activity, the associated termination property (compensatable, pivot, retriable) is also given.
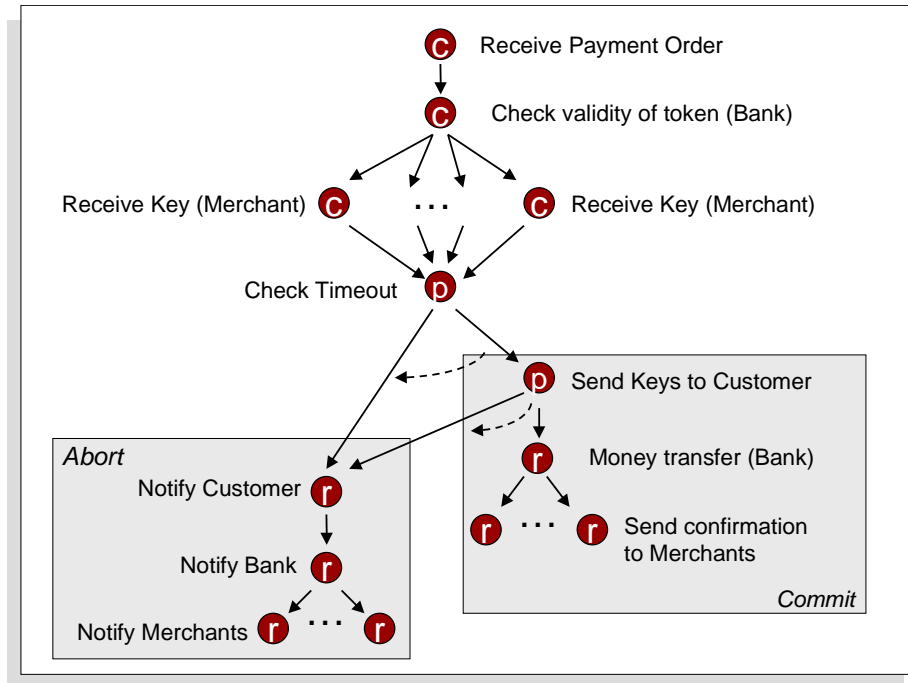
Figure 2: Structure of Payment Process

When a payment process is invoked, the customer first has to specify the payment information PI and all n bilaterally agreed order information (and thus also all different merchants) that have to be encompassed within one single payment transaction. Therefore, a tuple $(OI_c, M)_j$ with order information OI_c and merchant identifier M for each product j with $1 \leq j \leq n$ has to be sent to the Payment Coordinator

---

[1]Like in the traditional case, the customer has in the initial negotiation phase to agree upon the way the payment is processed with all merchants.

(*receive payment order*). Then, the value and validity of the payment information PI is checked (*check validity of token*). Given the validity of the payment information, the Payment Coordinator contacts all merchants, asks them to validate the order information $(OI_c)_j$ and in the case of successful validation, collects for each product j the key needed for decryption (*receive keys*). When all keys arrive within a given period of time (*check timeout*)[2], the Payment Coordinator sends all keys to the customer, sends a money transfer order to the bank in order to credit the merchant's accounts, and sends a confirmation about the successful termination of the payment to all merchants (*commit* of payment). Otherwise —when the customers view on the order information $(OI_c)_j$ and the merchants view $(OI_m)_j$ do not match for some j, when some keys are not available, when the timeout is exceeded, or when the validation of the payment information PI fails— no exchange will take place (*abort* of payment) but appropriate notifications are sent to all participants.

Based on the precedence and preference orders as well on the termination properties of each activity, it can be shown that this transactional payment process is correctly defined and thus provides guaranteed termination. Furthermore, it has to be shown that by all correct terminations, the desired semantics of atomic payment interactions (with respect to all three dimensions of atomicity) is provided. To this end, all possible executions have to be considered. Whenever some failure occurs prior to the termination of the *check timeout* activity, all previously executed steps are semantically compensated by sending a notification about the failure of the payment process to all participants (since this notification is also sent to the customer, she does not lose her payment information but can spent it later within other payments). After the successful transfer of the keys to the customer, the payment process is also terminated correctly since the real-world money transfer has previously been ensured by the bank (in the *check validity of token* step). Finally, when the transfer of keys to the customer fails (e.g., since she cannot be contacted), also appropriate notifications are sent to all participants and no real-world money transfer takes place (again, the payment information can be used by the customer for further payments).

This transactional payment process now provides money atomicity, certified atomic delivery and distributed purchase atomicity simultaneously. Since it is guaranteed that the payment information is only transferred in real-world money flow when the process terminates correctly and since no merchant receives this payment information directly, the customer is able to spent it again in the abort case of a payment process without being accused of double-spending. For certified atomic delivery, the same arguments as given in [CHTY96] hold: the Payment Coordinator persistently stores process information and is thus in the case of customer complaints able to verify whether the order information matches the goods delivered. Finally, since the process only terminates correctly when all merchants agree to commit, distributed purchase atomicity is also provided.

Aside of atomicity, also anonymity of the customer and provability have been identified as security aspects of payment protocols. Transactional payment processes do not provide total anonymity (since the Payment Coordinator needs to contact the customer in order to transfer the keys needed to decrypt all goods), but at least they provide partial anonymity. The customer may hide her identity (e.g., the IP address of the host she is using) to the merchants by applying anonymizing techniques (such as, for instance, [Ano99]). In order to hide the identity of the customer to the bank when issuing electronic cash token, cryptographic blinding techniques [CFN88] can be applied. Since the Payment Coordinator stores all process information (including the order information) persistently, the proof of the participation of a customer in a transaction and the service ordered in this transactions is possible (total provability).

By executing payment processes by a centralized Payment Coordinator, the monitoring of the state of a payment interaction is facilitated compared to the distribution found in current payment protocols. However, all participants (and especially the customer) have to trust this centralized Payment Coordinator. But since in the case of these payment processes only information about the merchants involved in a deal and the prizes of goods is available to the Payment Coordinator but no information about the single goods, this is equivalent to the amount and kind of data credit card organizations collect when customers perform payments with their credit cards.

---

[2]This activity only generates a log entry making the decision persistent; although it can technically be compensated, it is treated as pivot since compensation of the process is no longer allowed.

5 Conclusion

This paper provides a detailed analysis of requirements participants in E-Commerce payment impose with respect to atomicity issues. Different levels of atomicity can be identified which, however, are not simultaneously provided by existing approaches. Using the notion of processes, it has been shown that all payment interactions can be embedded into a single payment process where all possible levels of execution guarantees can be provided while at the same time the prerequisites of the participants are reduced. Finally, by applying the ideas of transactional process management, it has been shown how a Payment Coordinator supporting atomic and provable payment processes can be developed.

This process-based Payment Coordinator is currently being implemented within the WISE system [AFH$^+$99]. Based on this implementation, we will in our future work extend the analysis of payment processes to further properties (such as, for instance, anonymity, scalability, or flexibility). Our goal is to decouple these properties, to identify the building blocks needed to realize them and to flexibly generate payment processes with user-defined properties by plugging together the building blocks needed.

References

[ABFS97] G. Alonso, S. Blott, A. Feßler, and H.-J. Schek. Correctness and Parallelism in Composite Systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'97)*, Tucson, Arizona, May 12-15 1997.

[AFH$^+$99] G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt, and N. Weiler. WISE: Business to Business E-Commerce. In *Proceedings of the 9th International Workshop on Research Issues in Data Engineering. Information Technology for Virtual Enterprises (RIDE-VE'99)*, pages 132–139, Sydney, Australia, March 1999.

[Ano99] Anonymizer.com, 1999. `http://www.anonymizer.com`.

[CFN88] D. Chaum, A. Fiat, and M. Naor. Untraceable Electronic Cash. In *Proceedings of Advances in Cryptography (CRYPTO'88)*, pages 319–327. Springer, 1988.

[CHTY96] J. Camp, M. Harkavy, D. Tygar, and B. Yee. Anonymous Atomic Transactions. In *Proceedings of the 2nd Usenix Workshop on Electronic Commerce*, pages 123–133, November 1996.

[CTS95] B. Cox, D. Tygar, and M. Sirbu. NetBill Security and Transaction Protocol. In *Proceedings of the 1st USENIX Workshop on Electronic Commerce*, pages 77–88, July 1995.

[Dig99] DigiCash, 1999. `http://www.digicash.com/`.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[HS98] A. Hermanns and M. Sauter, editors. *Management–Handbuch Electronic Commerce*. Vahlen, 1998. In German.

[MRSK93] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A Transaction Model for Multidatabase Systems. *Bulletin of the Technical Committee on Data Engineering*, 16(2), June 1993.

[MV96] MasterCard and Visa. *Secure Electronic Transaction Specification*. MasterCard and Visa, draft edition, June 1996. Book 1: Business Description, Book 2: Programmer's Guide, Book 3: Formal Protocol Specification (Slightly revised version of Book 3 appeared August 1, 1997).

[MWW98] P. Muth, J. Weissenfels, and G. Weikum. What Workflow Technology can do for Electronic Commerce. In *Proceedings of the EURO-MED NET Conference*, 1998.

[SAS99]   H. Schuldt, G. Alonso, and H.-J. Schek. Concurrency Control and Recovery in Transactional Process Management. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'99)*, pages 316–326, Philadelphia, Pennsylvania, USA, May 31-June 2 1999.

[Sch98]   B. Schmidt. *Elektronische Märkte – Merkmale, Organisation und Potentiale*. In: [HS98]. 1998. In German.

[SSA99]   H. Schuldt, H.-J. Schek, and G. Alonso. Transactional Coordination Agents for Composite Systems. In *Proceedings of the 3rd International Database Engineering and Applications Symposium (IDEAS'99)*, pages 321–331, Montréal, Canada, August 1999.

[Tyg96]   D. Tygar. Atomicity in Electronic Commerce. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, pages 8–26, May 1996.

[Tyg98]   D. Tygar. Atomicity versus Anonymity: Distributed Transactions for Electronic Commerce. In *Proceedings of the 24th Conference on Very Large Databases (VLDB'98)*, New York, USA, August 1998.

[ZNBB94]   A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *Proceedings of the ACM SIGMOD Conference*, pages 67–78, 1994.

Performance Assessment and Configuration
of Enterprise-Wide Workflow Management Systems
(Extended abstract)

Michael Gillmann[1], Jeanine Weissenfels[1], Gerhard Weikum[1], Achim Kraiss[2]

[1]University of the Saarland
Department of Computer Science
P.O.Box 15 11 50
D-66041 Saarbrücken
{gillmann,weissenfels,weikum}@cs.uni-sb.de
http://www-dbs.cs.uni-sb.de/

[2]Dresdner Bank AG
Organization and Information Technology
IT Research and Standards
Jürgen-Ponto-Platz 1
D-60301 Frankfurt a.M.
achim.kraiss@dresdner-bank.com
http://www.dresdner-bank.com/

1 Introduction

The main goal of workflow management systems (WFMS) is to support the efficient, largely automated execution of business processes. Large enterprises demand the reliable execution of a wide variety of workflow types. For some of these workflow types, the availability of the components of the underlying, often distributed WFMS is crucial; for other workflow types, high throughput and short response times are required. However, finding a configuration of the WFMS (e.g., with replicated components) that meets all requirements is a non-trivial problem. Moreover, it may be necessary to adapt the configuration over time due to changes of the workflow load, e.g., upon adding new workflow types. Therefore, it is not sufficient to find an appropriate initial configuration; it should rather be possible to reconfigure the WFMS dynamically. The first step towards a dynamic configuration tool is the analysis of the WFMS to predict the performance and the availability that would be achievable under a new configuration, and to determine the best configuration for the current workflow load.

In this paper, we present an analytic approach that considers the performance as well as the availability of the WFMS in its assessment of the quality of a given configuration of a distributed WFMS. The approach is based on well known stochastic methods [Nel95, STP96, Tij94] and shows the suitability of these models to a new application field. The presented combination of the methods allows an analytic assessment of WFMS eliminating the usual time- and cost-intensive trial-and-error practice for system configuration. Particularly, we are able to rank the performance and the availability of different configurations which use replicated workflow servers. Moreover, we can predict the performance degradation caused by transient failures of servers. These considerations lead to the notion of performability [STP96], a combination of performance and availability metrics. Likewise, we are able to calculate the necessary number of workflow server replications to meet the specified requirements for performance and availability. So, a crucial part of a configuration tool for distributed WFMS becomes analytically tractable and no longer depends on trial-and-error practices or the subjective intuition of the system administration staff.

Although the literature includes much work on scalable WFMS architectures, there are only few research projects that have looked into the quantitative assessment of WFMS configurations with regard to performance and availability [DKO+98]. [BD99] presents several types of distributed WFMS architectures and discusses the influence of different distribution methods on the network and workflow server load, based on simulations. In [BD97], the sustainable throughput of a distributed WFMS is in-

creased by assigning subworkflows to appropriate workflow servers, based on online statistics about network partitions, network load, and expected communication costs. [SNS99] presents simple heuristics for the allocation of workflow type and workflow instance data onto servers. Work on WFMS availability has been presented in [HA98, KAG+96] that discuss how to efficiently increase the availability of process support systems by using standby mechanisms that allow a backup server to take over in the case of server failures.

The rest of the paper is organized as follows. In Section 2, we present our model of a distributed WFMS. In Sections 3 and 4, we develop a performance model and an availability model. In Section 5, we combine both models into the performability model that allows us to predict the influence of transient failures on the overall performance. Section 6 concludes the paper with a summary and an outlook on ongoing work.

2   System model of distributed WFMS

In this section, we describe a generic model of enterprise-wide distributed WFMS. Although the chosen system model is simple, it is powerful enough that we are able to capture the architecture models of most WFMS products and research prototypes in a reasonable way. Based on this model, we will introduce the central notions of the state and the configuration of a distributed WFMS. Finally, we present a model that stochastically describes the behavior of a single workflow instance.

Typically, distributed WFMS execute workflow instances in a partitioned and distributed manner, i.e., the workflow instance is partitioned into several subworkflows, which are executed in a distributed way on different workflow engines (e.g., with one workflow engine per partition/subworkflow type). These workflow engines typically run on several server machines distributed across an Intranet or even the Internet. Moreover, services that are "imported" from external companies can be integrated into the WFMS as subworkflows, and the WFMS of such a provider merely becomes another kind of workflow engine with a specific interface. The communication is handled by separate components, such as object request brokers (ORBs), other modules are responsible for workflow-specific functions like worklist management or monitoring, and the runtime state data of workflows is often stored in a DBMS. Finally, applications that are invoked from workflow activities may be spawned on dedicated application servers. All these components will be viewed as abstract servers of specific types in our system model.

2.1   Workflow server model

In workflows like the above examples, several cooperating components of a distributed WFMS are involved in the execution of a single workflow instance. We refer such to each component as a *server type*. For scalability and availability reasons, nearly all WFMS, both products and research prototypes, provide the replication of (performance-critical) server types within the system. Note that we have so far not said anything about the hardware (i.e., the server machines) that the workflow servers run on. It is possible (and often favorably or even unavoidable) to start workflow servers of different server types on the same server machine.

Figure 1 shows an example of the workflow server model. The dotted arcs indicate service requests between the several server types. But also within a single server type, workflow servers can request services from each other. For example, the execution of a subworkflow by a workflow engine that is not the same as the engine of the parent workflow is a service request, too. In our model, the communication services are only provided by some kind of *communication servers* such as object request brokers (ORBs) or TP monitors. The other server types do not communicate directly with each other but only via a communication server. In Figure 1, this is indicated by the solid lines.

In our system model, every server type s has a failure rate μ_s and a repair rate α_s (i.e., restart speed after a

failure). For simplicity, we assume that the time between two successive failures of a server type as well as the time to repair one workflow server are exponentially distributed with the expected values $1/\mu_s$ and $1/\alpha_s$, respectively.

2.2 Configuration of a distributed WFMS

With the presented workflow server model at hand, we are now able to define the central notion of the system configuration of a distributed WFMS.

Because of failures and repairs of workflow servers, the number of available workflow servers of one server type varies over time. For a given point of time, we call the vector $(X_1, X_2, ..., X_N)$ of the numbers of currently available workflow servers of each workflow server type the current *system state* of the WFMS. The initial system state of the WFMS, i.e., the system state with all workflow servers available, is called the *system configuration* of the WFMS.

The goal of this work is to build a configuration tool, based on the above system model, that is able to derive the best system configuration for a given workflow load. The configuration tool should aim to optimize the ratio of performance and cost or availability and cost, respectively, or even the combination of both, the performability. We will discuss our approaches to these three kinds of goals in the following sections. As all such optimizations depend on the (probable) behavior of workflow instances, we first need to introduce an appropriate model for the control flow of a workflow instance.

2.3 Stochastic modeling of control flow

For predicting the expected load induced by the execution of a workflow instance, we have to be able to predict the control flow of workflow instances. As workflows include conditional branches and loops, the best we can do here is to describe the execution stochastically.

A suitable stochastic model for describing the control flow of a simple workflow instance without sub-workflows is the model of continuous-time, first-order Markov chains (CTMC) [Nel95, STP96, Tij94]. A CTMC is a process that proceeds through a set of states in certain time periods. Its basic property is that the probability of entering the next state within some time only depends on the currently entered state, and not on the previous history of entered states. The mathematical implication is that the residence time in a state - that is, the time the process resides in the state before it makes its next transition -
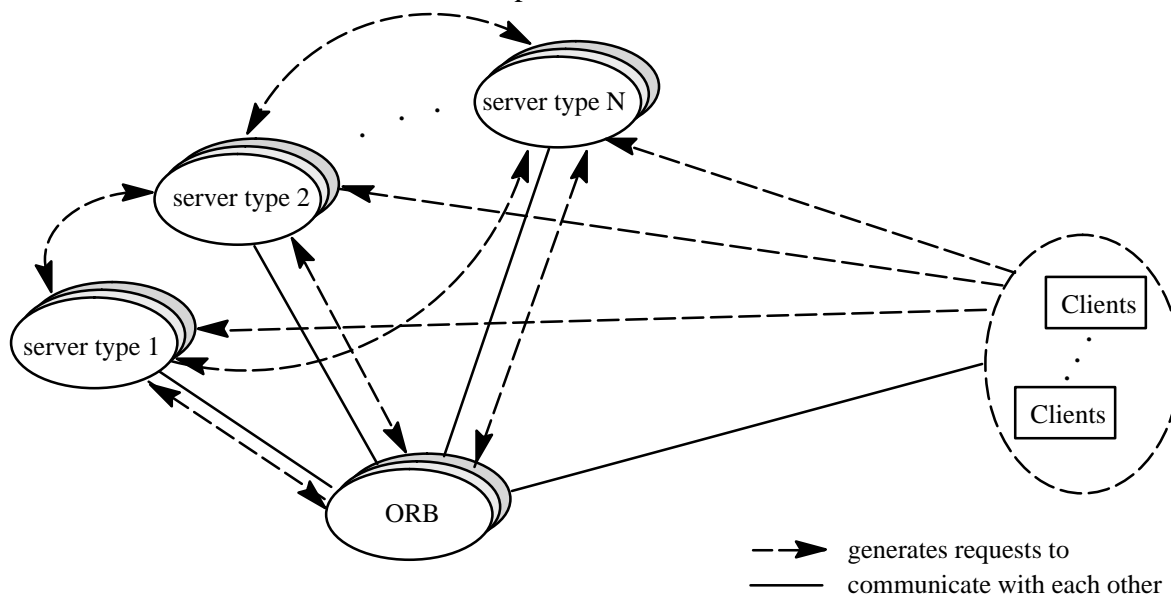

Figure 1: **Server model of an enterprise-wide distributed WFMS**

follows a (state-specific) exponential distribution. Consequently, the behavior of a CTMC is uniquely described by a matrix $P = \left(p_{ij} \right)$ of transition probabilities between states and the mean residence times H_i of the states.

Let $\{a_i \mid i = 1..n\}$ be the set of n activities being part of a workflow type X. Let X be a workflow type without any subworkflows. The impact of subworkflows can be handled recursively and will be explained later. The control flow of an instance of X will be modeled by a CTMC where the states correspond to the workflow activities a_i. The state transition probability p_{ij} corresponds to the probability that a workflow instance of workflow type X starts activity a_j after it has completed activity a_i. The transition probabilities have to be explicitly specified by the workflow designer based on the semantics of the conditions between the workflow activities or observed from real-life business processes. The mean residence time H_i corresponds to the mean turnaround time of activity a_i and also needs to be estimated at workflow specification time. Finally, we add an artificial absorbing state A to the CTMC. This state represents the point when the workflow instance represented by the CTMC is terminated; the residence time of state A is infinity. Furthermore, we add a transition from every state of the CTMC that represents a termination activity of the workflow instance into state A with the transition probability 1.

For workflow types with subworkflows, the subworkflows are initially represented by single fictitious states, i.e., each subworkflow is represented by a single state within the CTMC of the parent workflow. When workflows include parallelism, the parallel paths of the control flow are defined as subworkflows. In this case, the fictitious state represents all parallel subworkflows at once. As the mean residence time of the fictitious state, we use the maximum of the mean time until termination of all subworkflows within the fictitious state.

We derive the mean time until termination of a workflow type by the transient analysis of the CTMC representing the workflow type [Tij94]. In our model, a workflow instance of a workflow type terminates when the corresponding CTMC makes a transition into the absorption state A. So, the time until termination of a workflow type is equivalent to the mean time until the CTMC makes the first transition into state A. With the CTMC at hand, we are able to predict the expected load of workflow instances during their execution [Tij94] as shown in the following section.

3 Performance model

In this section, we present a performance model for a complete WFMS. We show how to describe the load for each workflow server type induced by the execution of a single workflow activity. We use this and the results from the transient analysis of CTMC presented in Section 2 for predicting the load induced by the execution of a entire workflow instance. Finally, we show how to predict the expected performance, i.e., sustainable throughput and expected response times of service requests, of the WFMS with a given system configuration.

3.1 Modeling activity-specific load

The execution of a workflow instance leads to the execution of a set of workflow activities. The execution of a workflow activity leads to the generation of service requests to different server types. Typically, the invocation of an activity leads to some initialization and termination load induced exactly once during the execution of the activity and to operational load induced continuously during the whole execution time of the activity. Therefore, we differentiate between the two following kinds of service requests.

- *Lump requests*. Lump requests are generated exactly once during the execution of the workflow activity. For example, if the activity corresponds to editing a text document, the activity generates

a number of lump requests for loading the text document, updating a database to reflect the changed workflow state, etc.

- *Operational requests.* During the execution of an activity, operational requests are generated with a specific rate. For example, these are generated by the exchange of messages between workflow engines for synchronisation and migration of workflow instances, saving of intermediate versions of the currently processed documents, etc.

In the following, the matrix (L_{sa}^t) denotes the number of lump requests being generated for workflow server type s when workflow activity a is invoked during the execution of an instance of workflow type t. The matrix (N_{sa}^t) denotes the generation rate of operational requests for workflow server type s during the execution of activity a.

3.2 Predicting the load induced by a single (sub-)workflow instance

To calculate the workflow load that one workflow instance generates on the several server types, we use the transient analysis of the CTMC presented in Section 2. To be exact, we combine the equivalent normalized CTMC [Tij94] and the already presented load matrices (L_{sa}^t) and (N_{sa}^t) to a Markov reward model (MRM). The feature of a MRM is that there are rewards for every state of the CTMC. To get the expected number of service requests by a single workflow instance, we calculate the expected reward earned until absorption [Tij94]. Let the matrix (L_{sa}^t) of lump requests and the matrix (N_{sa}^t) of rates of operational request be given for a workflow type without any subworkflows. Then, the expected number of service requests an instance of the workflow type t generates at server type s is given by

$$ r_{s,t} = \frac{1}{\nu_t} \left[\sum_{a \neq A} N_{sa}^t \sum_{z=0}^{\infty} p_{\emptyset a}^t(z) + \sum_{a \neq A} \sum_{z=0}^{\infty} p_{\emptyset a}^t(z) \sum_{b \neq A, b \neq a} q_{ab}^t L_{sb}^t \right], $$

where ν_t is the maximum of the departure rates of the states of the CTMC representing workflow type t, q_{ab}^t is the transition rate from state a to state b, and $p_{\emptyset a}^t(z)$ is the taboo probability that the process will be in state a after z steps without having visited the absorbing state A (starting in the initial state $\emptyset$).

The mean runtime R_t of an instance of a (sub-)workflow of type t is given by the mean time that the CTMC needs to enter the absorbing state for the first time, the so called first-visit-time of state A, and can be calculated by solving a system of linear equations [Tij94].

3.3 Incorporation of subworkflows

The expected number of service requests generated by an instance of a workflow type including subworkflows can be calculated recursively. For every state of the CTMC that represents a subworkflow or a set of parallel subworkflows, the entries L_{sx}^t and N_{sx}^t within the matrices (L_{sa}^t) and (N_{sa}^t) represent the lump requests and the rate of operational requests of the set x of nested subworkflows. We approximate the number of lump requests L_{sx}^t for a server type s by the sum of the expected number of service requests generated by the parallel subworkflows.

$$ L_{sx}^t = \sum_{y \in x} r_{s,y} $$

The rate of operational requests N_{sx}^t is set to 0 for every server type s.

3.4 Incorporation of multiple active workflows

By Little's law, the steady-state number of active instances N_{active}^t of workflow type t is given by the product of the arrival rate λ_t of new instances of type t and the mean runtime R_t of a single workflow of type t.

$$ N_{active}^t = \lambda_t R_t $$

The server-type-specific request arrival rate of a single instance of workflow type t is given by dividing

the expected number of service requests to server type s, $r_{s,t}$, by the mean runtime of an instance of t. We obtain the server-type-specific request arrival rate $l_{s,t}$ of all instances of workflow type t by multiplying $r_{s,t}$ with the mean number of active workflow instances.

$$l_{s,t} = N_{active}^t \frac{r_{s,t}}{R_t} = \lambda_t r_{s,t}$$

Finally, the request arrival rate l_s to workflow server s induced by all active instances of all workflow types is given by

$$l_s = \sum_t l_{s,t}.$$

3.5 Predicting response times for service requests

For predicting the mean response time of service requests, we model every server type as a set of k M/G/1 queueing systems where k is the number of server replications of the server type. We assume that the arriving service requests are uniformly distributed over all server replications. We thus compute the mean arrival rate of service requests for each M/G/1 queue by dividing the mean arrival rate of service requests for the server type by the number of servers k of the server type. The mean service time of service requests and the second moment of the service time distribution are parameters that can be estimated by online monitoring.

4 Availability model

In this section, we describe our availability model. It is based on the workflow server model described in Section 2. Based on this model, we analyze the influence of transient component failures on the availability of the WFMS.

Our availability model is again based on Continuous Time Markov Chains (CTMC). Here, every state of the CTMC represents a possible system state of the WFMS. A system state of the WFMS is modelled as an n-tuple with n being the number of different server types and each entry of the tuple representing the number of available workflow servers of a server type at one point of time. For example, the state *(2,1,1)* means that the WFMS consists of three different server types and there are *2* workflow servers of type *1*, *1* workflow server of type *2*, and *1* workflow server of type *3* currently available (the others have failed and are being restarted). When a workflow server of type i fails, the CTMC performs a transition to the state where the corresponding value for server type i is decreased by one. For example, the state $(X_1, ..., X_j, ...)$ is left when a workflow server of type j fails, and the state $(X_1, ..., (X_j - 1), ...)$ is entered. Analogously, when a workflow server of type i completes its restart, the value for server type i is increased in the target state of the firing transition. The failure rates and the repair rates of the server types are the corresponding transition rates of the CTMC. Note that non-exponential failure or repair rates (e.g., anticipated periodic downtimes for software maintenance) can be accommodated by refining the corresponding state with non-exponential residence time into a chain of exponential states [Tij94]. With the CTMC at hand, we are able to calculate for every state of the WFMS its steady-state probability by solving a system of linear equations [STP96]. From these probabilities, we can then derive the probability distribution of the number of available workflow servers for each server type, and thus the server type's steady-state availability.

5 Performability model

In this section, we briefly sketch a performability model that allows us to predict the performance of the WFMS with the effects of temporarily non-available servers (i.e., the resulting performance degradation) taken into account.

Our performability model is a hierarchical model constituted by a Markov reward model (MRM) for the availability CTMC and the performance model presented in Section 3. The probability of being in a specific state of the WFMS is given by the availability model presented in Section 4. As state-specific rewards, we use a function that assigns to every state of the availability CTMC the mean response time of service requests of the WFMS in the current state. The steady-state analysis of the MRM delivers the expected value for the response time of service requests for a given configuration of the WFMS [STP96].

6 Summary and Outlook

In this paper, we have discussed three models to derive quantitative information about performance, availability, and performability of distributed workflow management systems (WFMS) configurations. These models form the core towards an assessment and configuration tool for enterprise-wide, large scale WFMS. We are in the process of implementing such a tool. The tool consits of four components: *mapping* of workflow specification onto the presented models, *calibration* by means of statistics from monitoring the system, *evaluation* for given input parameters, and the computation of *recommendations* with respect to specified administration goals. When the tool is to be used for configuring a completely new workflow environment, most input parameters have to be inetellectually estimated by a human expert. Later, after the system has been operational for a while, these parameters can be automatically adjusted, and the tool can the make appropriate recommendations for reconfiguring the system. For a first evaluation of our overall approach, we have defined a WFMS benchmark [GMW+99] and we are conducting measurements of various products and prototypes, including our own Mentor-lite system. These measurements will serve as a first yardstick for the accuracy of our performance assessment model.

References

[BD97] T. Bauer, P. Dadam, A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration, 2nd IFCIS Conf. on Cooperative Information Systems (CoopIS), Charleston, South Carolina, 1997

[BD99] T Bauer, P. Dadam, Distribution Models for Workflow Management Systems - Classification and Simulation (in German), Technical Report, University of Ulm, Germany, 1999

[DKO+98]   A. Dogac, L. Kalinichenko, M. Tamer Ozsu, A. Sheth (Eds.), Workflow Management Systems and Interoperability, NATO Advanced Study Institute, Springer-Verlag, 1998

[GMW+99] M. Gillmann, P. Muth, G. Weikum, J. Weissenfels, Benchmarking of Workflow Management Systems (in German), 8th German Conf. on Database Systems in Office, Engineering, and Scientific Applications (BTW), Freiburg, Germany, 1999

[HA98] C. Hagen, G. Alonso, Backup and Process Migration Mechanisms in Process Support Systems, Technical Report, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1998

[KAG+96] M. Kamath, G. Alonso, R. Günthör, C. Mohan, Providing High Availability in Very Large Workflow Management Systems, 5th Int'l Conf. on Extending Database Technology (EDBT), Avignon, France, 1996

[Nel95] R. Nelson, Probability, Stochastic Processes, and Queueing Theory, Springer-Verlag, 1995

[STP96] R. A. Sahner, K. S. Trivedi, A. Puliafito, Performance and Reliability Analysis of Computer Systems, Kluwer Academic Publishers, 1996

[SNS99] H. Schuster, J. Neeb, R. Schamburger, A Configuration Management Approach for Large Workflow Management Systems, Int'l Joint Conf. on Work Activities Coordination and Collaboration (WACC), San Francisco, California, 1999

[Tij94] H.C. Tijms, Stochastic Models, John Wiley and Sons, 1994

Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows

Thomas Bauer, Peter Dadam

University of Ulm, Dept. of Databases and Information Systems

{bauer, dadam}@informatik.uni-ulm.de, http://www.informatik.uni-ulm.de/dbis

Abstract

In large workflow management systems (WfMS), it is particularly important to control workflows (WF) in an efficient manner. A very critical factor within this context is the resulting communication overhead. For this reason we have developed an approach for distributed WF control, which tries to keep the communication overhead low. In this paper, this approach is described and examined by means of a simulation.

1 Introduction

Enterprise-wide and cross-enterprise WF scenarios are characterized by a large number of users and many concurrently active WF instances. Therefore, the WF servers have to cope with a high load in total. Furthermore, in such an environment, the different organizational units (OU) are often far away from each other and connected by slow wide area networks (WAN). For this reason, the load of the communication system is an extremely critical aspect. Because of the resulting communication overhead a centralized WF control is often not applicable (at least not at reasonable costs). Another reason is that the WF systems used are often very heterogeneous which makes a centralized WF control rather complicated if not even impossible. In the ADEPT project[1] we, therefore, have developed an approach for distributed WF control which addresses these issues.

In the next section, some approaches for distributed WF management are presented and the distribution model of ADEPT is described. In Section 3 the different distribution models are compared by means of a simulation. Section 4 discusses related work and Section 5 concludes with a summary and an outlook on future work.

2 Distribution Models

In this section, different approaches for distributed WF management are presented and an appropriate model for enterprise-wide and cross-enterprise usage is developed.

---

[1] ADEPT stands for <u>A</u>pplication <u>D</u>evelopment <u>B</u>ased on <u>E</u>ncapsulated <u>P</u>re-Modeled Process <u>T</u>emplates.

2.1 Distribution of Entire Workflows

The simplest approach for distributed WF management is to control a WF instance always completely by one WF server. This can be done, e.g., by distributing the WF control to WF servers by WF type. That is, whenever a WF instance of type x is started, it will be completely controlled by the WF server responsible for workflows of this type. By doing so, the total load is divided up among the WF servers. This approach works well if almost all actors performing the activities of one WF belong to the same OU.

2.2 Partitioning of Workflows

If the actors of the different activities of a WF are geographically located far away from each other, the distribution model described above generates high communication costs. The reason is that all activities of a WF instance are controlled by the same server and, therefore, have to communicate with this server (even if this server is far away). In such cases it would be favorable, if for each activity a WF server which is closely located (at best in the same subnet) to the actor of the activity could be used.

To achieve this goal, the WF is partitioned and each *partition* is assigned to that WF server which is located next to the potential actors of the activities belonging to it (c.f. Fig. 1). At run-time, when the control moves from one partition to the next, a *migration* becomes necessary: The current state of the WF instance (WF control data and the values of the data elements[2]) is transferred to the WF server of the subsequent partition. Subsequently, this WF server takes over the control of the WF instance. Migrations are not for free, however. They also cause communication costs and contribute to the total load of the WF servers. In ADEPT, therefore, migrations are used only if they are profitable in the sense that they improve the total communication behavior. For example, in most cases it does not make much sense to migrate a WF instance to another server and back again just for the execution a single activity which is performed by an actor of another OU. These two migrations cause higher costs than to control the activity by an unfavorable server. For a WF designer, it is difficult to decide which are the most appropriate server assignments for the activities. For this reason, ADEPT supports the WF designer by sophisticated build-time components, which calculate those server assignments which will lead to a minimization of the total communication costs at run-time (see [BD97]).
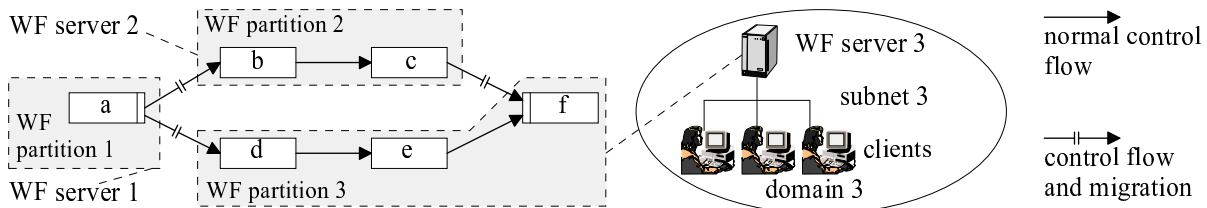
Figure 1 Partitioning of a WF schema and distributed WF execution.

---

[2] The WF schema information itself is completely replicated at all WF servers. Therefore, only the relevant instance information has to be transferred to the target server. Opposed to the WF schema, WF instance information is not replicated or stored centrally in order to avoid synchronization overhead or (long distance) communication during the "normal" execution of WF activities (i.e., execution without migration).

2.3 Variable Server Assignments

The approach described in Section 2.2 is reaching its limiting factors if a WF contains *dependent actor assignments* (c.f. activities 2, 3, and 4 in Fig. 2). Consider the case where a hospital has several wards of the same type and where a patient is (more or less arbitrarily) transferred to one of these wards after his admission in the emergency room. The subsequent activities 2, 3, and 4 (c.f. Fig. 2) will then only be performed by the medical staff of this ward. At build-time, however, it is not known which actor and thus which ward will be selected in activity 1 (at best, probability considerations are possible). This means, in turn, that no suitable (static) server assignments for this WF can be determined.
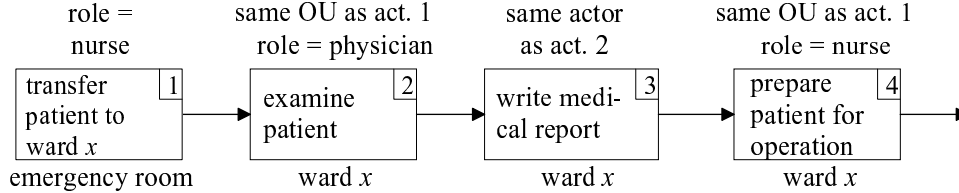
Figure 2 Example of a WF with dependent actor assignments.

In principle, problems of this kind could be solved by determining the server assignments fully dynamically at run-time. I.e., after the completion of each activity a computation to determine the most appropriate WF server for the subsequent activity or activities takes place. In doing so, one would (basically) achieve the optimal distribution, since at this point in time most information is available for this decision. Unfortunately, this approach is not feasible because of performance reasons, in general. At run-time the WF servers have already to manage a high workload and should therefore not be burdened with additional (and partially rather complex) computations to determine the optimal distribution.

Variable server assignments [BD98b, BD99a] are a compromise between static server assignment at build-time and dynamic server selection at run-time: At build-time, a logical server assignment expression is determined which only has to be evaluated at run-time. In the example shown in Fig. 2, the server assignments for the activities 2, 3, and 4 result as: "server in the domain[3] of the actor performing activity 1". After the completion of activity 1, its actor and because of that also the OU (i.e. ward) for the actors who shall perform the activities 2, 3, and 4 are known. Therefore, at this point in time, the WF server of the right OU can always be chosen to control these activities. This means that information is used which was not existing at build-time and also not at the point in time this WF instance was started. Therefore, this approach is more powerful than mere static server assignments.

In ADEPT the following server assignment expressions are used at present:

1. $ServAss_k = "S_i"$
   Server S_i is statically assigned to activity k (c.f. Section 2.2).
2. $ServAss_k = "Server(x)"$
   The activity k shall be controlled by the same server as activity x.
3. $ServAss_k = "Domain(Actor(x))"$
   Activity k is assigned to the server which is located in the domain of the user who has executed activity x.

---

[3] A domain is a subnet together with the corresponding WF server and clients.

3

4. $ServAss_k = "f(Server(x))"$ or $ServAss_k = "f(Domain(Actor(x)))"$
   A function f can be applied to server assignments of type 2 and 3.
5. $ServAss_k =$ any given expression, which does not correspond to type 1-4
   The WF designer may specify own server assignment expressions.

The ADEPT WfMS is supporting the WF designer in determining the most appropriate server assignment expression. At build-time, on request, the optimal (variable) server expressions of type 1 - 4 for this type of WF are computed. At run-time of a WF instance only these expressions have to be evaluated which can be done very efficiently. Therefore the load of the WF servers is only negligible higher as with static server assignments, but the communication costs are significantly reduced. Because of lack of space, it is not possible to describe the computation of the optimal server assignment expressions in this paper. The interested reader is referred to [BD98b, BD99a].

3 Evaluation

Dependent actor assignments occur in many application domains. Take the processing of a loan request at a bank, for example. Many steps of the examination of this request take place at the branch office of the customer. To compare the distribution models described in Section 2, we use a (simplified) clinical WF. The comparison is based on a simulation (for details and other simulations see [BD99b]). The actors belong to 7 OU. To each OU belongs one subnet (and one WF server in the distributed cases). The WF consists of a sequence of the following activities:
3 activities in the emergency room
1 activity to be performed by a ward physician (he transfers the patient to his ward 1-5)
5 activities to be performed by a physician of this ward x
1 activity to be performed in the laboratory
5 activities to be performed by a physician of ward x

The simulation presented subsequently is only used to compare the different distribution models. It was not the intention to detect overload situations. For this reason, even for the central case, it is assumed that the WF server is not overloaded. The data produced by the simulation are used to calculate the total load of the WF servers, the load per WF server, the load per subnet, and the load of the gateways. Fig. 3 shows a graphical representation of the result. The values are normalized in a way that the load of the central case results as 100.

3.1 Central WF Server

The load of the single WF server is defined as 100. The same applies to the average load per subnet and to the load of the gateways. The central WF server is located in exactly one subnet. This means that this subnet is burdened with the whole communication of this central WF server. Therefore, it represents a potential bottleneck. This irregular distribution of the load leads to the high load 387.1 of this subnet, whereas the other subnets are only burdened with 52.1 on the average. (That is, we have (as defined) an average load of 100 over all subnets, but have a peak of 387.1 in the subnet of the server.)
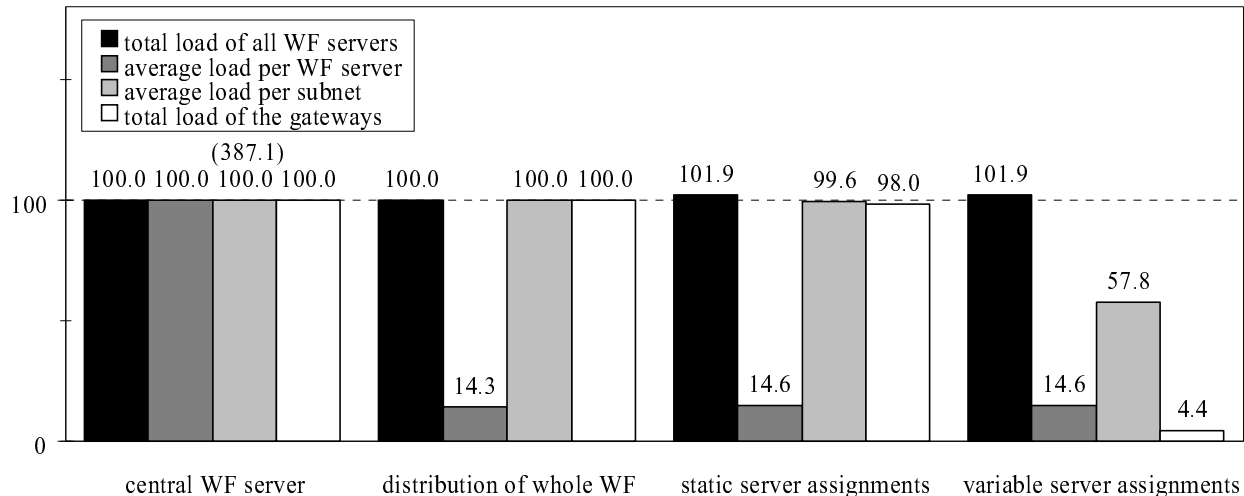
Figure 3 Results of the simulation of a clinical WF for different distribution models.

3.2 Distribution of Entire Workflows

If always entire WF are assigned to the WF servers, the total load of the WF servers is the same as in the central case because there are no migrations as well. By doing so, this load is divided up among the 7 WF servers of the WfMS, in principle[4]. Therefore, the load per WF server is 14.3, which is the best value of all distribution models. The load of the subnets and the load of the gateways are identical to the central case because in both cases the server of ward 1 is used. The bottleneck in the subnet of the WF server, however, does not exist any more, since different WF types may be controlled by different WF servers.

3.3 Static Server Assignments

If the WF is partitioned, migrations become necessary (e.g., there is a migration from the WF server of the emergency room to the WF server of the selected ward). This results in a higher total load of the WF servers. This load, however, is shared among several WF servers. The load of the subnets and the load of the gateways are reduced because appropriate WF servers are chosen for the partitions. Since static server assignments are hardly suitable for the WF considered, the improvements are very small.

3.4 Variable Server Assignments

Variable server assignments allow to control activities performed by a physician of ward x by the WF server of this ward. Because of this, the load of the subnets and the load of the gateways can be reduced significantly. The load of the subnets is almost halved (57.8). Nearly all the communication between WF server and client can now be handled within one subnet instead of two subnets (the subnets of the

---

[4] In this simulation, however, one WF server was burdened with the whole load, because we have simulated the execution of instances of only one WF type.

WF server and the client). Gateway communications occur very seldom (4.4) because variable server assignments always allow to select the WF server in the appropriate ward.

A goal of ADEPT is to reduce the communication load. The simulation has shown that this can be achieved by the use of variable server assignments. A disadvantage is that the total load of the WF servers increases due to the migrations. The use of additional WF servers, however, can compensate this effect.

4 Related Work

This section gives an overview of different approaches for distributed WF management. Due to lack of space, the concepts beyond these approaches are only briefly mentioned. A more detailed discussion can be found in [BD98a, BD98b, BD99b]. Some research prototypes (e.g., Panta Rhei [EG96], WASA [WHKS98]), which do not primarily consider scalability issues, and most of the commercial WfMS use a central WF server. Another extreme is a completely distributed system (Exotica/FMQM [AMG$^+$95], INCAS [BMR96]). The machine of the user currently performing an activity also controls the WF instance. Therefore, there is no need for any WF server.

There are several multi-server approaches: In METUFlow [Dog97] the WF instances are controlled in a distributed manner. It is not discussed, however, how the location for a WF server is chosen. The Exotica/Cluster approach [AKA$^+$94] and MOBILE [HS96] assign WF servers to whole WF instances. In addition, MOBILE enables remote WF servers to control subprocesses [SNS99]. MENTOR [MWW$^+$98], WIDE [CGS97], CodAlf, BPAFrame (both [SM96]) and METEOR$_2$ [DKM$^+$97] use WF partitioning. The partitions are statically assigned to WF servers. In addition, CodAlf and BPAFrame use a trader to select one of the assigned WF servers at run-time. The TEAM Model [Pic98] discusses the treatment of cooperations between autonomous enterprises. Activities are (statically) assigned to the WF server of the corresponding enterprise. To our best knowledge, ADEPT is the only approach that uses variable server assignment expressions.

5 Summary and Outlook

In order to avoid overloading of the WF servers and of the communication network, distributed WF control in enterprise-wide application scenarios is indispensable. This requires partitioning of WF schemas and migrations. The communication behavior can be further improved if variable server assignment expressions are used. These expressions can be determined at build-time, allow the selection of a suitable WF server to keep most of the communication local within the same subnet, and require almost no additional effort at run-time.

In cross-enterprise WF applications [Pic98], the activities shall be controlled by the WF server of the enterprise the activities "belong" to, in most cases. For this reason, if different activities of the same WF are performed by actors of different enterprises, then partitioning of WF and migrations are required. If an activity may be performed by actors of more than one enterprise, then static server assignments are not adequate. With variable server assignments, it is possible to assign such an activity to a WF server depending on WF instance data; i.e., the WF server may belong to different enterprises. The decision – which enterprise performs and controls a specific activity – may depend on previous activities (e.g., the actor of the activity "select job") or it may depend on WF data (e.g., the data element "contractor").

In addition to scalability and performance aspects, the treatment of heterogeneity is important for the support of enterprise-wide and cross-enterprise WfMS. This problem must be solved by defining appropriate standards for the interoperability of (heterogeneous) WfMS. What kind of functionality should be offered in order to adequately support such scenarios under communication aspects has been discussed in this paper. Certainly, a lot of other aspects, like transaction support [Ley97] to guarantee the robustness of the WfMS or the possibility to adapt running WF instances [RD98, RBD99] to deal with exceptional cases are also very important.

References

[AKA$^+$94] G. Alonso, M. Kamath, D. Agrawal, A. El Abbadi, R. Günthör, and C. Mohan. Failure Handling in Large Scale Workflow Management Systems. Technical Report RJ9913, IBM Almaden Research Center, 1994.

[AMG$^+$95] G. Alonso, C. Mohan, R. Günthör, D. Agrawal, A. El Abbadi, and M. Kamath. Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management. In *Proc of the IFIP Working Conf. on Information Systems for Decentralized Organisations*, Trondheim, 1995.

[BD97] T. Bauer and P. Dadam. A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration. In *2nd IFCIS Conf. on Cooperative Information Systems*, pages 99–108, Kiawah Island, SC, 1997.

[BD98a] T. Bauer and P. Dadam. Architekturen für skalierbare Workflow-Management-Systeme – Klassifikation und Analyse. Ulmer Informatik-Berichte 98-02, Universität Ulm, 1998.

[BD98b] T. Bauer and P. Dadam. Variable Migration von Workflows in ADEPT. Ulmer Informatik-Berichte 98-09, Universität Ulm, 1998.

[BD99a] T. Bauer and P. Dadam. Variable Migration von Workflows und komplexe Bearbeiterzuordnungen in ADEPT. Ulmer Informatik-Berichte, Universität Ulm, 1999. (to appear).

[BD99b] T. Bauer and P. Dadam. Verteilungsmodelle für Workflow-Management-Systeme – Klassifikation und Simulation. Ulmer Informatik-Berichte 99-02, Universität Ulm, 1999.

[BMR96] D. Barbará, S. Mehrotra, and M. Rusinkiewicz. INCAs: Managing Dynamic Workflows in Distributed Environments. *Journal of Database Management*, 7(1):5–15, 1996.

[CGS97] S. Ceri, P. Grefen, and G. Sánchez. WIDE – A Distributed Architecture for Workflow Management. In *7th Int. Workshop on Research Issues in Data Engineering*, Birmingham, 1997.

[DKM$^+$97] S. Das, K. Kochut, J. Miller, A. Sheth, and D. Worah. ORBWork: A Reliable Distributed CORBA-based Workflow Enactment System for METEOR$_2$. Technical Report #UGA-CS-TR-97-001, Department of Computer Science, University of Georgia, 1997.

[Dog97] A. Dogac et al. Design and Implementation of a Distributed Workflow Management System: METUFlow. In *Proc. of the NATO Advanced Study Institute on Workflow Management Systems and Interoperability*, pages 61–91, Istanbul, 1997.

[EG96] J. Eder and H. Groiss. Ein Workflow-Managementsystem auf der Basis aktiver Datenbanken. In J. Becker, G. Vossen, editor, *Geschäftsprozeßmodellierung und Workflow-Management*. International Thomson Publishing, 1996.

[HS96] P. Heinl and H. Schuster. Towards a Highly Scaleable Architecture for Workflow Management Systems. In *Proc. of the 7th Int. Workshop on Database and Expert Systems Applications*, pages 439–444, Zurich, 1996.

[Ley97] F. Leymann. Transaktionsunterstützung für Workflows. *Informatik Forschung und Entwicklung, Themenheft Workflow-Management*, 12(2):82–90, 1997.

[MWW+98] P. Muth, D. Wodtke, J. Weißenfels, A. Kotz-Dittrich, and G. Weikum. From Centralized Work-flow Specification to Distributed Workflow Execution. *Journal of Intelligent Information Systems*, 10(2):159–184, 1998.

[Pic98] G. Piccinelli. Distributed Workflow Management: The TEAM Model. In *Proc. of the 3rd IFCIS Int. Conf. on Cooperative Information Systems*, pages 292–299, New York, 1998.

[RBD99] M. Reichert, T. Bauer, and P. Dadam. Enterprise-Wide and Cross-Enterprise Workflow-Management: Challenges and Research Issues for Adaptive Workflows. In *Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI (Informatik '99)*, Paderborn, October 1999.

[RD98] M. Reichert and P. Dadam. ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[SM96] A. Schill and C. Mittasch. Workflow Management Systems on Top of OSF DCE and OMG CORBA. *Distributed Systems Engineering*, 3(4):250–262, 1996.

[SNS99] H. Schuster, J. Neeb, and R. Schamburger. A Configuration Management Approach for Large Workflow Management Systems. In *Proc. of the Int. Joint Conf. on Work Activities Coordination and Collaboration*, San Francisco, 1999.

[WHKS98] M. Weske, J. Hündling, D. Kuropka, and H. Schuschel. Objektorientierter Entwurf eines flexiblen Workflow-Management-Systems. *Informatik Forschung und Entwicklung*, 13(4):179–195, 1998.

Most of our publications are available at the following URL: http://www.informatik.uni-ulm.de/dbis/papers

Modelling inter-organizational processes with process model fragments

Frank Lindert

Wolfgang Deiters

Fraunhofer Institute Software and Systems Engineering

Joseph-von-Fraunhofer-Str. 20

44227 Dortmund

{deiters,lindert}@do.isst.fhg.de

Abstract

Today most of the workflow management approaches focus on centralized business processes that are being carried out within one orgaization.However, business practice demand that more and more inter-organizational processes have to be considered. Therefore we developed an approach for a decentralized process management basing on the concept of process model fragments. The goal of the approach is especially to consider the autonomicy of the organizations that participate in inter-organizational processes. A process model fragment is related to an organization and describes the part of the process the organization (resp. its human agents) is responsible for. By interconnecting process model fragments the information exchange between the organizations is described. The concept that is proposed in this paper is generic an can be adopted to many workflow management approaches.

1. Introduction

Within the last years workflow management has become a technology that is being more and more used in order to support business processes. Based on an enactable description of the processes, so called workflow model, the processes are being supported by workflow systems that usually interpret the processes, assign to the various people involved in the process the tasks they have to perform, and, provide the tools and objects that are needed to perform the tasks. Thus workflow systems drive and monitor the business processes.

Various approaches have been developed for managing business processes. What is common to these approaches is that they nearly all focus on the management of what we call centralized business processes, processes that run in one organization mainly at one geographical location. However, there are several arguments showing that a centralized approach turns out to be inappropriate for managing many of the larger scale industrial processes:

- Processes run across different geographical locations
  Various processes are performed in companies that are spread over different geographical locations. Coordinating these processes from one central point makes the complete workflow management application dependent on the connections between the organizations. A breakdown of the central system or the telecommunication lines between the locations would turn down the complete process in all locations.

- Processes run across different organizational units or even different cooperating partners
  In different application areas, such as for example in the automotive industry, business processes are spread over different organizations or even completely independent companies that cooperate in order to produce a common product or service. These partners operate independent from each other on their own „local" process parts, furthermore they have agreed on process interfaces on a contractual basis. Since (partial) autonomicy is

an important goal for the different partners a central process description would not be accepted among them. Furthermore, autonomicy is very important in virtual organizations and in the area of e-commerce especially in business-to business scenarios.

- Heterogeneous workflow systems are being used
  Different partners that work together in a common process partially have their individual workflow system installed and running. Thus an approach that enables an interoperation of heterogeneous workflow applications (i.e. applications coming from different vendors) is needed.

Deriving from these requirements we developed a decentralized process management approach. This approach starts from the assumption not to concentrate the process information of the various cooperating partners in one model but rather to develop individual process models for the different cooperating partners. We will call these individual process models process model fragments in the following. The fragments are the process definitions of the (partially) autonomously operating partners that have to be fitted with interfaces in order to arrange the overall coordination.

Within this paper we are going to briefly sketch the concept of process model fragments and the interconnection of these fragments towards decentralized process models. This generic concept is part of the concept of fractal process management which is described in [Lind99]. Beside the process modelling the whole concept considers also organizational aspects, a fractal process life cycle and a system architecture which considers the components of a fractal process management including security issues.

2. Process model fragments

The previous section gave a brief motivation for the development of a decentralized process management approach dealing with inter-organizational processes. To support these processes we need a concept capable to deal with organizational frontiers, autonomous organizations[1] that participate in the process, and, heterogeneous environments. The main idea behind that is to extend process management towards managing cooperating partners each of which staying independent from the others as much as possible (e.g. in virtual communities). Thus the aspects of autonomy of partners and coordination among partners have to be balanced well. The notion of autonomy that we use here is derived from [Warn95] addressing vitual, self-contained, self-organizing units that pursue own goals.

The basic idea of our approach is to build up process fragments that are as independent as possible from each other. A process fragment has interfaces that are used to connect the fragments together. These connected process fragments represent the inter-organizational process. Each process fragment is related to an organizational unit that is responsible for the fragment. The fragment contains all activities the organizational unit has to perform during enaction of the process. The organizational unit can autonomously describe the fragment and enact the fragment.

There exist several approaches that focus on the subdivision of a given process into pieces. For example, in the Exotica/FMQM project [AMGA95] the subdivision of a process is based on a formerly completely described process model. The basic idea there is to build sets of activities that have to be performed at the same "place" (place is the server, the responsible users are connected with). Each set of activities is transferred to the appropriate server. In [GrGr95] a process model is subdivided into pieces that afterwards can be extended. In [NSH98] the subdivision is based on the hierarchical decomposition of a process. An activity of a process can be refined by a another process. During enaction each process is enacted on one of the available enaction servers. [BaDa97] focuses on the distribution of the enaction of a process to optimize the available resources. [GAHM98] introduces a collaborative editing approach to describe distributed software processes. All these approaches do not consider the "autonomy" of the organization that participate in the process. [LuWh99] use gateways to support cross-organizational workflow management. This approach considers the privacy of the organizations which is similar to the autonomy we want to support but they focus on the system architecture but does not consider synchronization aspects, complex documents types and the description of interfaces in the process model.

To support inter-organizational processes the process fragment approach has to consider the following requirements:

1. Each participating organizational unit has to be able to model, analyse and enact its part of the process autonomously. Then the organizations can hide their processes from other organizations. In virtual environments the

---

1. In the following we use the term organization to describe both, organizational units in large organizations and organizations as a whole.

internal business processes are one key competence of the organizations they want to preserve from the other organizations.[1]

2. Each process fragment interacts with other fragments. Like a process a process fragment needs a set of input documents it works on and produces a set of output documents. The input documents are provided by other process fragments and the output documents are used by other process fragments. However the passing of inputs and outputs between fragments is not restricted towards a „procedure interface". We rather have to distinguish between start-up inputs and termination outputs, i.e. documents that are necessary inputs for starting a fragments and documents that are produced as result of the fragment and intermediate inputs / outputs, documents that are being exchanged during the fragments operation.

3. Each participating organization uses its own process management approach. Therefore the process fragment approach must not focus on one specific process management approach. It should be generic to be implemented in various existing process management approaches.

.
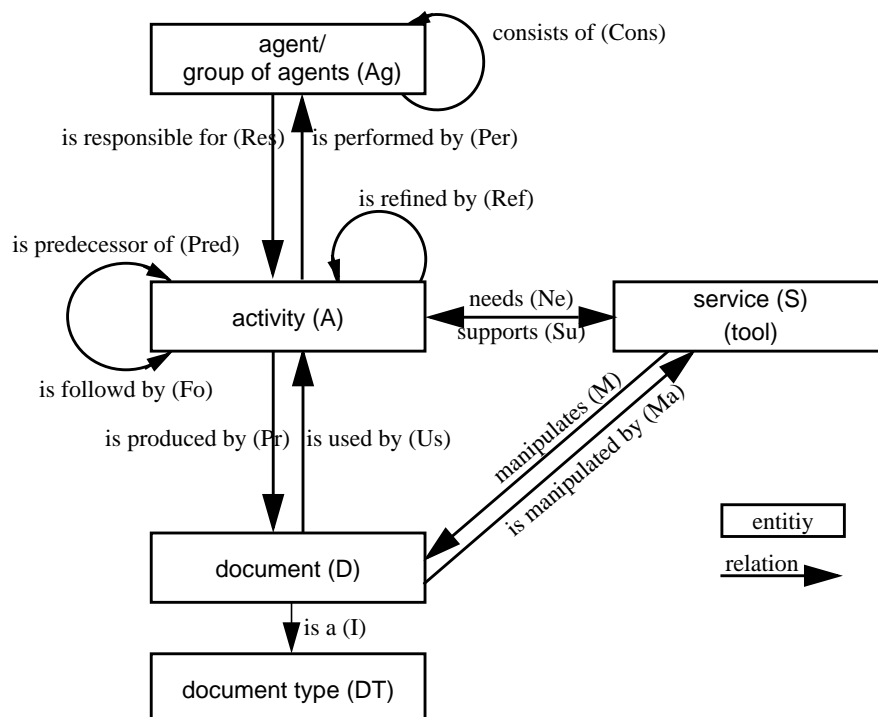
figure 1: General metamodel for process models

To fulfil theses requirements we introduce the concept of process fragments. They are described by extended process models. The extension describes on the one hand the interface, the process fragment uses to interact with other fragments and on the other hand provides additional information for the analysis and enaction of the process fragments. The interface description is public. It is used by other organizations to connect process fragments. The sequence of activities of a process fragment is private to the organization. It is not published to other fragments.

The development of the approach is based on the general metamodel in figure 1. This metamodel has been derived from the models presented in [Deit93] and [Jabl95]. Additionally we consider services and agents. So opposite to the general process description languages like WPDL [WfMC97], PSL [MSID98], PIF [LGJM96], CPR [PeCa97] this metamodel does not describe all aspects of a process model, e.g. transition rules or complex organizational structures of the agents.

The following process model fragment definition is based on the view based FUNSOFT [DeGr98] approach and the aspect based approach MOBILE [JB96]. These approaches subdivide the description of a process model into

1. The assumption for the model development is as follows: The cooperating partners meet in an initial design phase and thereby agree upon the overall process model goal, the main interfaces, etc. Then, the model definition of the individual fragments takes place autonomously. Interfaces are checked and made fitting. Thus, the meaningfulness of the whole process model is achieved by the process model development lifecycle (see [Lind99]).

different views (aspects). Each view focuses on one aspect of the description. Examples of views are structural view (functional aspect, behavioural aspect, information aspect), service view (operational aspect) or the project management view (organizational aspect).

We define a process model as follows:

Definition 1: Process model[1]:

$$PM \;=\; (StV, SV, IV, OV) \text{ with}$$

$$StV \;=\; (A, Pred, Fo, Ref, Pr, Us) \qquad \text{is the structural view,}$$

$$SV \;=\; (S, Ne, Su, M, Ma) \qquad \text{is the service view,}$$

$$IV \;=\; (D, DT, I) \qquad \text{is the information view and}$$

$$OV \;=\; (Ag, Cons, Per, Res) \qquad \text{is the organizational view.}$$

The subdivision of the process into process fragments that we focus on within this paper is related to the control flow and the data flow. For this we have extended the structural view in order to describe the interfaces of a process fragment. The control flow describes the sequence of activities and the data flow describes the documents the activities consume and produce. In an inter-organizational process we have inter-fragment control flow and inter-fragment data flow. Both have to be described in the process model fragment.

To describe the inter-fragment control flow we introduce a new entity in the metamodel called events. This is necessary because the control flow ((Fo) and (Pred) in figure 1 are relations between activities. Since we want to hide the activities of a process fragment a description serving as a substitution is needed. An event can be produced or consumed by activities. In figure 2 the description of inter-fragment control flow is shown in an example.
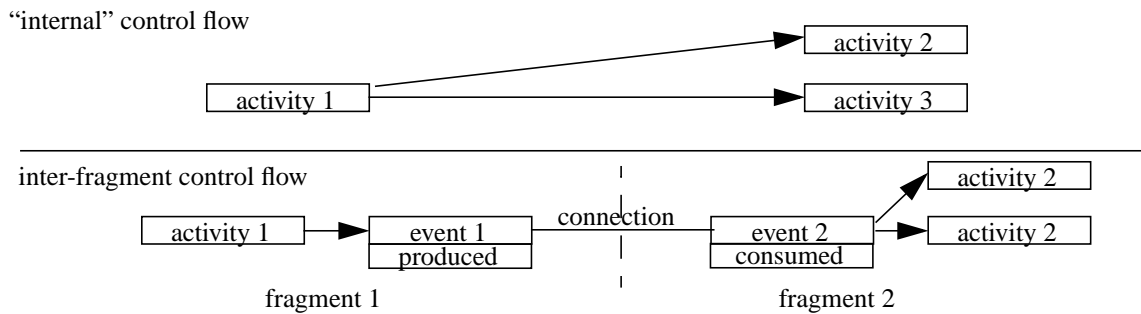
figure 2: description of inter-fragment control flow with events

To integrate the events in the metamodel we define a new entity and two new relations between the entity "event" and the entity "activity".
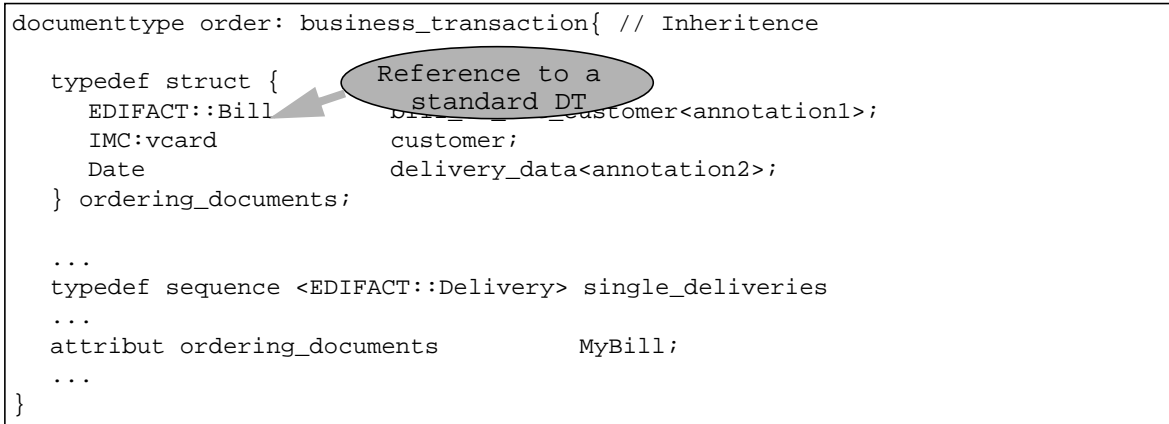
To describe the inter-fragment data flow no new entities are needed. There is rather the task to handle the proper transfer of documents (incl. document type mapping) between the different fragments[2]. Therefore, the documents have to be described in detail in the interface description in order to ensure that the receiver of a documents gets the document he expected. The documents that are passed between activities in the process fragment have no influence on the inter fragment data flow. The main problem is to describe the structure and the semantics of a document on both sides. The sender has to describe what documents he will provide and the receiver has to describe what documents he needs. In several application domains there exists standards that describe documents, e.g. CDIF, STEP, EDIFACT. These standards describe the structure and the semantics of documents. In other application domains no standards exist.

In order to describe the documents in the inter-fragment data flow we introduce an external document type description language (DTDL) which is based on the OMG IDL. We removed the method descriptions of IDL and inserted a mechanism to consider the existing standards. Due to the limited space in this paper we only give a short example

---

1. The abbreviations in the definition can be taken from figure 1.
2. Of course, document types also have to be described within the fragments. However, this is being done in the languages of the approaches that model the resp. fragments and, thus, no subject of discussion for the scope of this paper.

of a DTDL document description:

```
documenttype order: business_transaction{ // Inheritence

   typedef struct {                 Reference to a
      EDIFACT::Bill       standard DT customer<annotation1>;
      IMC:vcard           customer;
      Date                delivery_data<annotation2>;
   } ordering_documents;

   ...
   typedef sequence <EDIFACT::Delivery> single_deliveries
   ...
   attribut ordering_documents         MyBill;
   ...
}
```

figure 3: Example of a document type definition with DTDL

Similar to the document type problem there are other aspects to be described in order to define what a fragment expects and what a fragment offers:

- Number of documents.
  Whenever more than one document, e.g. a set of documents, has to be transferred, a minimum and a maximum number of documents has to be described. Some approaches use containers of documents to describe the data flow, e.g. the Petri-net based FUNSOFT [DeGr98] approach. Other approaches describe each document with one entity in the process model like LinkWorks.

- Copy or move.
  In some cases it is necessary to specify that a copy of a document has to be transferred and the original document remains in the sending fragment.

- Optional or required.
  Sometimes not all documents are expected. Some may be optional, e.g. for a credit request in case of a large credit amount some optional reports have to be transferred.

- Fragment identifying document.
  Sometimes it is necessary to determine the receiving fragment (see below). In this case a fragment identifying document can help to do this. For example, an application can identify the appropriate process fragment with the application number.

- Informal annotation.
  Due to the fact that the semantics of a document cannot be described any time we consider an informal, multi-media annotation to describe further information of a document.

Summing up the general metamodel has to be extended by new entities, relations and by attributes in order to cope with process fragments. The extended metamodel is shown in figure 4:
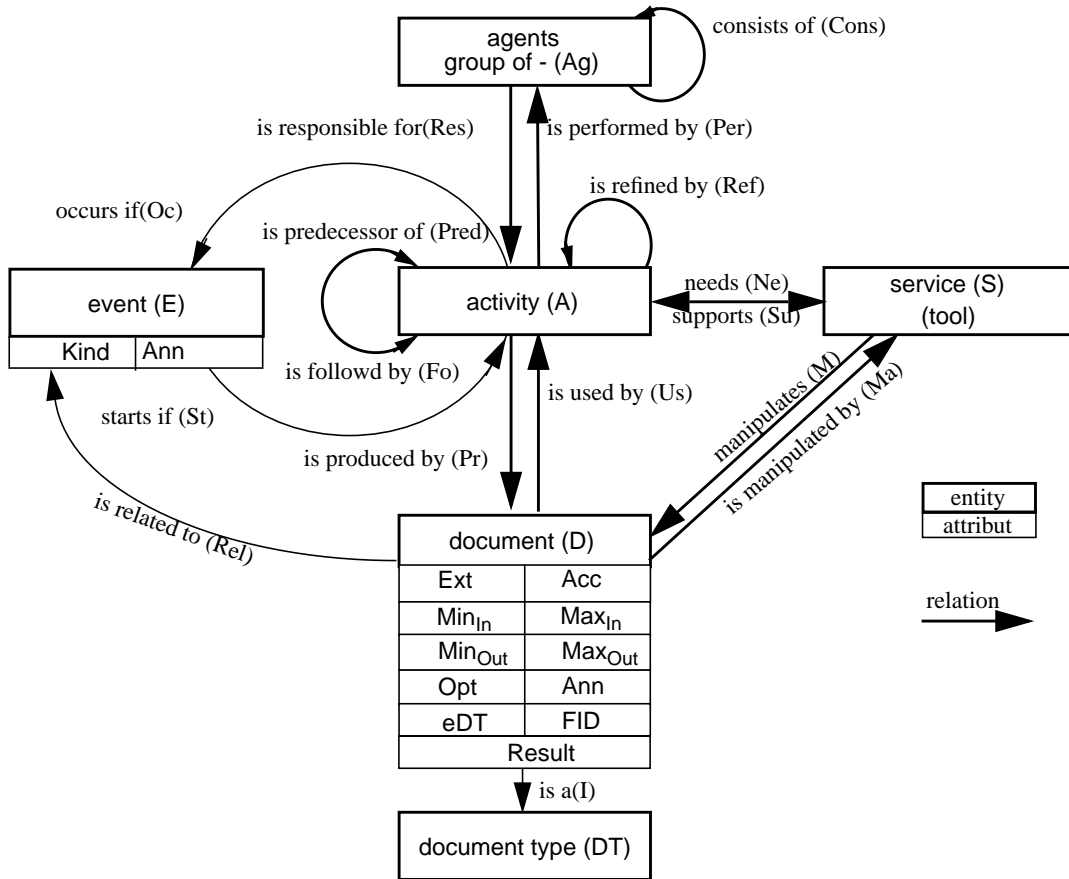
.

figure 4: Extended general metamodel

The process model fragment is defined as follows:

Definition 2: Process model fragment: $PMF_{OE} = (PM', ER, IV, Con)$ with

$PM' = (PV', SV, DV, OV)$ is a process model that consists of an extended process view PV' (extended by the relations (St) and (Oc)), a service view SV, a data view DV and a organizational view OV,

$IV = (E, Kind, Rel, Ext, Acc, Min_{In}, Max_{In}, Min_{Out}, Max_{Out}, Opt, Ann, eDT, Hor)$ is the new interface view that describes the interface of the process fragment[1], ER is the external representation of the process model fragment that describes the process model PM' in a simplified representation. With this external representation the organization can decide how much details about the internal activities are to be published. Furthermore during enaction the state of the process fragment can be queried. The external representation is completely independent from the process model PM'. Con is a structured set of connection points of the process model fragment. These connection points are used to connect the process model fragments. Each connection point consists of a set of events and external documents.

An inter-organizational process consists of several process fragments. So the description of the whole process consists of a set of process model fragments that have to be connected in order to describe the control flow and data flow between the fragments. Such a complete description does not need to exist at process model build time because sometimes not all fragment information (or even the fragments at all) are known at that point in time. Consider e.g. the complex process of developing and producing a new car. Such a process lasts a long time, sometimes several years. Even the organizations that participate in the production of the car are not known all at the beginning

---

1. The attributes FID (fragment identifying document) an Result (specifies a special kind of output document) is part of Hor, the description of horizontal connection points.

of the process. So is has to be possible to complete the process description even during its enaction. Therefore it is possible to leave interfaces unconnected at the start of the enaction. During run-time the process engine has to consider the interfaces and to notify the process modeller in case of an unconnected interface. Due to the fact that a description of the whole process often does not exist in advance only limited possibilities for analysis exist, e.g. deadlocks that are spread over several process fragments can't be computed[1]. This can't be avoided because otherwise the autonomy of the participants will be violated. Taking into account this autonomy requirement partially consistency and correctness has to be assured by introducing organizational rules in the process management lifecycle [Lind99].

There exist two major types of connections between interfaces of process fragments. On the one hand we consider vertical connections. This type of connection allows the hierarchical refinement of activities of a so called father process fragment by child fragments. It is a connection between an activity in the father fragment and a child fragment. This type of connection is similar to the approach in [NSH98] and to the hierarchical connection scenario of the WfMC [WfMC96]. We extend these approaches by the multi-vertical connections that allows one process fragment to be connected by more than one vertical connection (cf. figure 5, in this example we do not use events to describe the inter-fragment control flow)
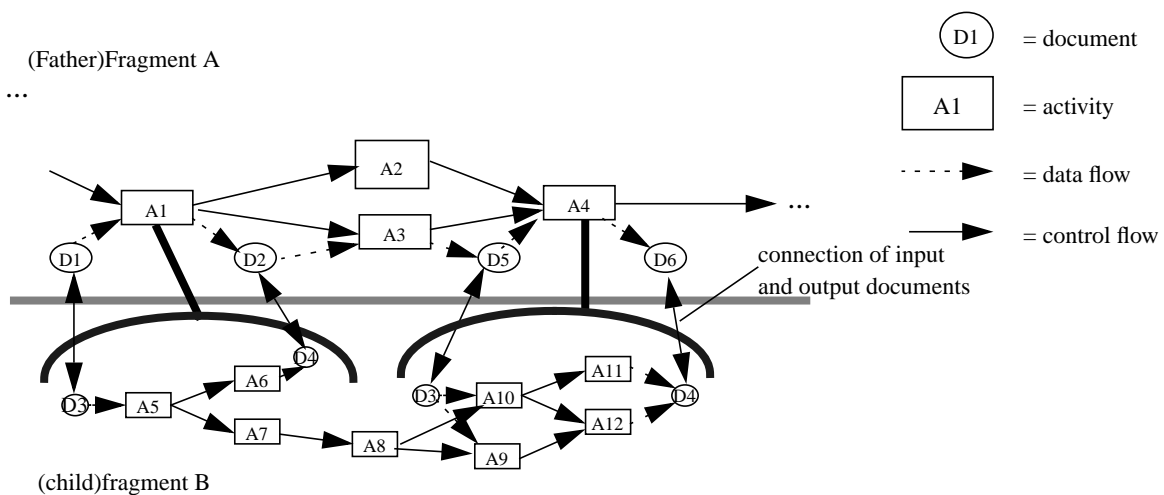
figure 5: Multiple vertical connection

The other type of connection is the horizontal connection. This type connects documents and events. By that the passing over of documents and events can be described. This kind of connection is similar to the discrete connection of the WfMC. In figure 6 an example of a horizontal description is shown.

---

1. In [Aals98] the verification of interorganizational workflow is discussed. This approach is based on petri nets. All process fragments have to be described by petri nets and the verification uses the whole petri nets so the autonomy requirement is not assured.

fragment 1 connection fragment 2 activity

D1
a

E_p1 relation E_k1

A1

D2
a

D4
e

A3

E_k1

D5
e

D7 A5

A4

local document

event fragment 3 control flow/data flow

A2 E_p2 E_K2 A6

Output document D3
a

D6
e

A7

Input document
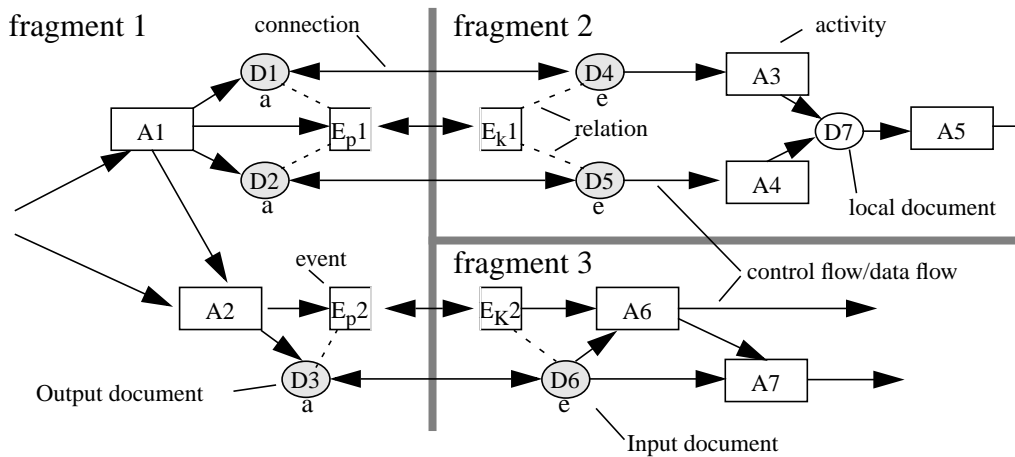
figure 6: Horizontal connection of three fragments

In the process model fragment we define the description of offered and expected connections using the relation Con. This relation is a set of alternative connections a process model fragment offers resp. expects. Due to the limited space in this paper we cannot describe all features of the connection process in more detail. However, the connection of the process model fragments is a negotiation process. The organizations that want to connect their process models have to ensure that the corresponding interfaces fit together. This negotiation process is supported by the attributes we introduced in the interface description. Based on these attributes we defined a set of consistency rules that have to be considered in order to specify valid connections. Each organization stores the connections locally. This is necessary to fulfil the autonomy requirements.

With the process model fragments each organization can autonomously describe its part of a inter-organizational process independent from the other participating organizations. Only the interface descriptions are published. So we fulfil the requirements we listed at the beginning of this section. The general metamodel is independent from any existing process model approaches, thus it becomes possible to implement it in different workflow systems.

3. Summary

With the concept of process model fragments inter-organizational processes can be described. The concept considers the autonomicy of the participating organizations. In this paper we described the modelling of inter-organizational processes. Furthermore we developed an architecture that extends existing process management systems in order to model, analyse and enact inter-organizational processes. First components of the architecture have been developed and implemented in the project VORTEL [BDFL96]. In this project we integrated the CORMAN System based on the FUNSOFT approach and the Systems LinkWorks and FlowMark focusing on the enaction of workflows. [Henn98] focused on an architecture for supporting a decentralized process modelling. He introduced various traders (based on the CORBA technology) that allow to find process models of cooperating partners that can be matched, that allow a negotiation on the interface, and, that allow to match object types of the interface of different fragments.

References

[Aals98]   Aalst, W.M.P., „Interorganizational Workflows“, in Proc. of PROLAMAT'98, p. 21-43, Trento, 1998

[AMGA95]   Alonso, G., Mohan, C., Günthör, R., Agrawal, D., El Abbadi, A., Kamath, M., "Exotica/ FMQM: A Persistent Message Based Architecture for Distributed Workflow Management", in Proc. IFIP WG 8.1 Working Conference on Information Systems for Decentralized Organizations, Trondheim, 1995

[BaDa97]   Bauer, T., Dadam, P., "A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration", in Proc. Second IFCIS Conf. on Cooperative Information Systems, Charleston, South California, 1997

[BDFLS96] M. Böhm, W. Deiters, M. Friedrich, F. Lindert; "Workflow Management as Teleservice", in Computer Networks and ISDN Systems, Vol. 28, No. 4, S. 1961-1969, 1996

[Deit93] Deiters W., „A View Based Approach to Software Process Management", University of Berlin, 1993

[DeGr98] Deiters, W., Gruhn, V., "Process Management in Practice - Applying the FUNSOFT Net Approach to Large-Scale Processes", in Automated Software Engineering 5, S. 7 - 26, Kluwer Automated Press, 1998

[GAHM98] Grundy, J.C., Apperley M.D., Hosking, J.G., Mugridge, W.B., "A Decentralized Architecture for Software Process Modeling and Enactment", in IEEE Internet Computing, September October, S. 53-62, 1998

[GrGr95] Graw, G. und Gruhn, V.; "Distributed Modeling and Distributed Enaction of Business Processes", In Software Engineering ESEC '95, 5th European Software Engineering Conference, Sitges, Lecture Notes in Computer Sience 989, Springer Verlag, S. 8-27, 1995

[Henn98] Henning, D., "Eine Systemarchitektur für dezentrales Prozeßmanagement in autonomen Organisationseinheiten", Diplomarbeit Universität Dortmund, 1998

[JB96] Jablonski S., Bussler C., Workflow Management. Modelling Concepts, Architecture and Implementation, International Thomson, 1996

[LGJM96] Lee, J., Gruninger, M., Jin, Y., Malone, T., Tate, A., Yost, G. et al, "The Process Interchange Format and Framework", Working Group, Version 1.1, 1996

[LuWh99] Ludwig, H., Whittingham, K., „Virtual Enterprise Co-ordinator - Agreement-driven Gateways for Cross-Organisational Workflow Management", Proc. of the Int. Joint Conference on Work Activities Coordination and Collaboration (WACC'99), San Francisco, 1999

[Lind99] Lindert, F., „Fraktales Prozeßmanagement" (in german), Phd. Thesis, Technical University of Berlin, 1999

[MSID98] Manufacturing Systems Integration Division, "The PSL Home Page", http://www.mel.nist.gov/psl/, 1998

[NSH98] Neeb, J., Schaumberger, R., Schuster, H., „Using distributed Object Middleware to implement scalable Workflow Management Systems", in Ozsu, T., Dogac, A., Ulusoy, O. (Eds.), Proc of the 3rd Biennial World Conf. on Integrated Design and Process technology, Berlin, 1998

[PeCa97] Peace, R. A., Carrico, T., „The JTF ATD Core Plan Representation: A Progress Report", Proc. of the AAAI Spring Symposium on Ontological Engineering, 1997

[Warn95] Warnecke, H.-J., „Die fraktale Fabrik - Revolution in der Unternehmenskultur" (in German), Springer, Heidelberg, 1995

[WfMC96c] Workflow Management Coalition, "Workflow Standard - Interoperability, Abstract Specification", Document Number WfMC TC-1012, Version 1.0, 1996

[WfMC97] Workflow Management Coalition, "Interface I: Process Definition Interchange", Document Number WfMC TC-1016, Draft 6.94, 1997

Adaptive Workflows based on Flexible Assignment of Workflow Schemas and Workflow Instances
– Extended Abstract –

Mathias Weske

Westfälische Wilhelms-Universität Münster

Steinfurter Straße 107, D-48149 Münster, Germany

weske@helios.uni-muenster.de

1   Introduction

Traditionally, workflow management deals with controlling the execution of application processes according to pre-defined specifications, known as workflow schemas [3, 8, 13]. This approach is well suited to support application processes with fairly static control structures. Real-world application processes, however, are not static in general. In contrast, they may require dynamic modifications to react quickly to new challenges imposed by the environment of the application process. While the exact definition of flexibility in workflow management systems is still under discussion [12, 7], it is widely accepted that dynamic modifications – or adaptations – of running workflows is an important feature of a flexible workflow management system [10, 2, 11]. This extended abstract sketches the conceptual design and implementation of dynamic modifications in the context of the WASA project at the University of Muenster.

2   The WASA$_2$ Approach

We use a workflow language based on process graphs, similar to those used by IBM's MQSeries Workflow (formerly IBM FlowMark). Workflows can be atomic or complex, there are data flow and control flow constraints between workflows, and technical and organizational information is ttached to workflow schemas. WASA$_2$ is based on an object-oriented approach: Workflow schemas and workflow instances are objects, which are characterized by a state and a behavior and which communicate with each other by sending and receiving messages. This general approach allows the flexible re-use of workflow schemas as sub-workflows in different complex workflow schemas, such that the embedding of the different occurrences of a workflow schema can be different with respect to its start condition and control flow and data flow constraints.

The object-oriented design and a distributed object middleware allow distributed workflow executions, such that workflow objects of a given workflow application can reside in different sites of a distributed computing system. In this case, workflows are controlled in a distributed manner without the need for a centralized workflow engine, which can become a performance bottleneck in large-scale workflow applications. In terms of flexibility, modeling workflow schemas and workflow instances as objects allows to change the association of a workflow instance with its controlling workflow schema. Hence, during its life time, a workflow instance can be controlled using different workflow schemas.
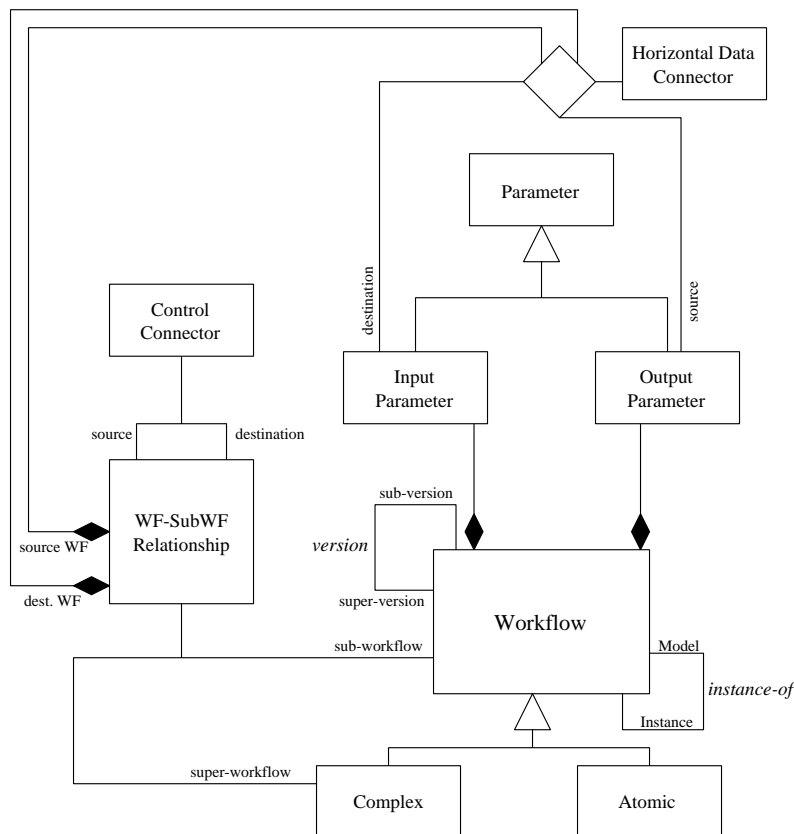
Figure 1: WASA$_2$ Workflow Meta Schema (Simplified Version).

The remainder of this section sketches the conceptual design of the system, which is specified in a workflow meta schema; a simplified version of the WASA$_2$ workflow meta schema is shown in Figure 1. Due to space limitations, we do not elaborate on different modeling alternatives for object-oriented workflow management systems [14], and we only discuss the parts of the workflow meta schema which are relevant for dynamic modifications.

The workflow class is in the center of the WASA$_2$ workflow meta schema; it contains workflow schema objects and workflow instance objects. Workflows can be either atomic or complex. The work-flow hierarchy (i.e., the relationship between a complex workflow and its sub-workflows) is modeled by the WF-SubWF Relationship class, which defines a relationship between a complex workflow and a workflow, which can be complex or atomic. Workflow schemas and workflow instances are identified by states. The relationship between a workflow instance and the respective workflow schema is represented by an instance-of relationship. Each workflow schema can be associated with multiple workflow instances, while each workflow instance is associated with exactly one workflow schema at any given point in time. This relationship allows the flexible assignment of workflow instances to workflow schemas, as will be discussed in more detail in the remainder of this extended abstract.

Other parts of the meta schema deal with control connectors and data connectors; a control connec-tor relates two WF-SubWF Relationship objects, defining execution order of the respective workflows. Each workflow has a set of input parameters and a set of output parameters, whose commonalities are represented in a Parameter class. Horizontal data flow represents data flow between workflows of a common super-workflow, while vertical data flow represents data flow between a super-workflow and

its sub-workflows. The complete WASA$_2$ workflow meta schema as well as the design of the system and its implementation based on CorbaServices is presented in [15].

3   Dynamic Modifications in WASA$_2$

The ability to dynamically modify the structure of running workflow instances is an important feature of a flexible workflow management system, since it allows running workflow instances to adapt to changes in the environment. In this context, "environment" refers to the market environment of processes, including new services provided by competitors, new and faster or more cost efficient ways to produce or deliver goods, and providing new services or parts thereof. Besides changes in the market, there may be new legal regulations that have to be implemented by application processes. For instance, consider a new legal regulation that a single checking mechanism has to be changed to a double-checking mechanism. As a consequence, workflow instances have to use the new checking policy in order comply with the new regulations. Changes in the technical environment of the process are another motivation for dynamic modifications. Assume there are new tools available to perform tasks more efficiently then the active and all future workflow instances should make use of the new infrastructure.

In dynamic modifications, it is important to define correctness criteria which determine if and when a workflow instance can be adapted to a new workflow schema. To motivate our correctness criterion, we start by discussing correctness in workflow applications in general, i.e., without dynamic modifications. Since in the workflow context, generic correctness properties like in database transaction processing (e.g., conflict serializability, recoverability) do not suffice to describe correct workflows, application specific correctness criteria have to be defined. From an application-oriented point of view, these criteria are specified in business process models, which describe which activities have to be performed, and what are the constraints between them. When supporting business processes by workflow technology, the correctness of workflow instances is specified in workflow schemas. Hence, a workflow execution is correct if and only if it satisfies the criteria specified in the respective workflow schema. Control flow and data flow constraints as well as role information and technical information are examples of properties which are specified in workflow schemas and which have to be met by workflow instances. The task of a workflow management system is to make sure each workflow instance is executed according to its workflow schema.

Based on this perception of correctness in the workflow context, the approach to controlling dynamic changes in WASA$_2$ is fairly simple, yet effective:

> *A workflow instance can be dynamically modified, i.e., it can be adapted to a new workflow schema, if the workflow instance could have been controlled from the beginning using the new workflow schema.*

For an example consider Figure 2, which shows a workflow schema S (a), a modified workflow schema S' (b) and two currently active workflow instances i and j, based on the original workflow schema S ((c) and (d), resp.). Notice that workflow instances i and j are correct with respect to workflow schema S, since they can be continued according to S. We now assume that there is a dynamic modification, changing workflow schema S to S', shown in Figure 2(b).

When a workflow administrator decides to change a workflow dynamically, he or she first suspends the execution of the workflow. This is necessary, since otherwise race conditions between the normal
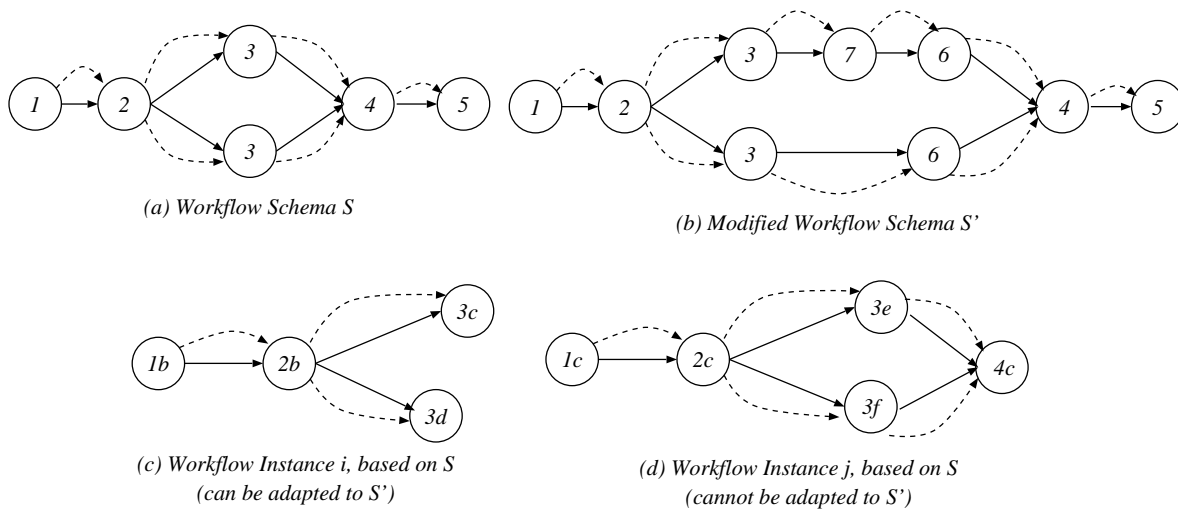
(a) Workflow Schema S

(b) Modified Workflow Schema S'

(c) Workflow Instance i, based on S
(can be adapted to S')

(d) Workflow Instance j, based on S
(cannot be adapted to S')

Figure 2: Workflow Schemas S and S' and Workflow Instances i, j, based on S.

workflow execution and the correctness checks would occur. In our example, given workflow instances i and j (so far based on S), the system now has to decide whether these can be continued with the modified workflow schema S'. This check is performed by analyzing the current state of the workflow instances and matching the structure of the new workflow schema S' against the states.

We first consider workflow instance i. It is obvious that the workflow instance can be adapted to S', since there is a mapping between the sub-workflow instance already executed on behalf of i and sub-workflow schemas of S'. More technically, the decision whether or not a workflow instance can be adapted to a new workflow schema is taken based on a mapping. The sub-workflow instances which have already been executed are mapped to the sub-workflow schemas of the new workflow schema. In our example, sub-workflow instance $1b$ of workflow instance i is mapped to sub-workflow schema 1 of workflow schema S', and sub-workflow instance $2b$ is mapped to sub-workflow schema 2 of S'. The two workflow schemas based on 3 can be mapped to $3a$ and $3b$. Since furthermore the control flow constraints of the workflow instance and the workflow schema comply, and assuming that data flow constraints also comply, a mapping can be found. As a consequence, i can be continued to become a complete workflow instance based on S': after the termination of $3c$, an instance $7a$ (based on workflow schema 7) and an instance $6a$ (based on workflow schema 6) can be started, and after the completion of $3d$ an instance $6b$ (based on workflow schema 6) can be executed. Finally, workflow instances based on workflow schemas 4 and 5 can be performed sequentially. Hence, the resulting workflow instance is correct and complete with respect to the new workflow schema S'.

Along the lines of this argumentation it is clear that j cannot be adapted to the new workflow schema. It proceeded further than i; in particular, it already started a sub-workflow instance based on workflow schema 4, i.e., $4c$. Since in the modified workflow schema S', 4 can only be stated after additional sub-workflow instances have completed, workflow instance j (d) violates this requirement. Even starting sub-workflow instances for 7 and 6 (as specified in S') right away would not help, since by the control flow constraints specified in S', an instance of 4 cannot start until workflow instances based on workflow schemas 7 and 6 have been executed. This constraint imposed by workflow schema S' cannot be satisfied by any continuation of j. Hence, j cannot be modified dynamically with respect to workflow schema S'.

4

In WASA$_2$, the instance-of relationship associates a workflow instance object with a workflow schema object. At each point in time, each workflow instance object is associated with exactly one workflow schema object, while each workflow schema object can be associated with an arbitrary number of workflow instance objects. Given this organization, dynamic modifications can be implemented by changing the respective instance-of relationship objects at runtime. To implement a dynamic modification based on a workflow schema S with numerous currently active workflow instances, the following steps are carried out:

- create a new workflow schema (or use an existing one) S'

- based on the instance-of relationship, let C be the set of all workflow instances which are associated with S

- compute a set $C' \subseteq C$ of workflow instances which can be adapted to the new workflow schema S', based on finding a mapping as sketched above

- allow a workflow administrator to select a subset of C' of workflow instances, which will actually be dynamically modified; update instance-of relationship of these workflow instances accordingly

- all workflow instances are continued using their respective instance-of relationships, i.e., modified workflow instances are continued with S', and non-modified workflow instances are continued with the original workflow schema S

To perform an adaptation of workflow instance i to S', sub-workflow instances which are no longer needed are deleted, and new sub-workflow instance objects are created, as specified in the new workflow schema S'. These workflow instance objects are embedded in the context of the complex workflow instance by creating the respective WF-SubWF Relationship objects. In our example, new workflow instance objects $7a$, $6a$, $6b$, $4d$ and $5b$ are created and attached to the complex workflow using WF-SubWF Relationship objects. The workflow is continued with the execution of the sub-workflow instances $7a$ and $6a$.

4   Related Work

Two recent workshops were devoted to adaptive and flexible workflow management [7, 12]. In [5] a taxonomy of adaptive workflow management is proposed. In particular, the constantly changing market environment of business processes is regarded as a major motivation for flexible workflow management. Process level adaptations and resource level adaptations are among the requirements for a flexible workflow management system. Techniques for exception handling in workflow management systems are identified and classified in [9, 1]. An approach to enhance the flexibility of workflow management systems based on an integration of workflow and workspace management techniques is discussed in [6]. To classify flexibility requirements, *a priori* and *a posteriori* flexibility is characterized by properties of the application that are known before it starts and after it has started, resp.

In [10] a workflow language ADEPT is proposed, which allows to specify workflow schema using symmetric graphs. There are different node types, reflecting for instance split and join nodes, and start and end nodes of loops. Based on this workflow language, a model ADEPT$_{flex}$ supporting a set of operations to change the structure of workflows is defined, allowing to dynamically change

workflows, while keeping their symmetric structure. ADEPT does not consider workflow schemas; only workflow instances are discussed. Hence, the approach does not consider dynamic changes in presence of multiple workflow instances based on a modified workflow schema. There is an operational running prototype implementing dynamic modifications according to $ADEPT_{flex}$. In [4], a Petri-Net based approach to model workflows which includes flexibility mechanisms is proposed. Simple dynamic modifications are allows, for instance to leave unspecified defined portions of the net to be filled when the workflow executes; this is denoted by late modeling; this functionality is implemented in the CORMAN prototype [4].

5   Conclusions

This extended abstract sketches the design of controlled dynamic modifications of workflow instances by a flexible assignment of workflow schemas to workflow instances. By allowing to change the assignment of workflow instances and workflow schemas at different points in time, different schemas can be used to control a workflow. An adaption of a workflow instance i to a workflow schema $\mathcal{S}$ can be done whenever i can be continued such that it fits $\mathcal{S}$. This elegant yet simple characterization of the correctness of a dynamic modification allows to maintain the correctness property of workflow instances with respect to workflow schemas, while providing workflow flexibility by dynamic modifications. Future work in the WASA project will be centered around user interface design, and we plan to use the WASA$_2$ system in real-world workflow applications which make use of the capabilities of the system.

References

[1] Deiters, W., Goesmann, T., Just-Hahn. K., Lffeler, T. Rolles, R.: *Support for exception handling through workflow management systems.* In Proceedings CSCW-98 Workshop: Towards Adaptive Workflow Systems. (downloaded from http://ccs.mit.edu/klein/cscw98/paper19 on 09-24-1998)

[2] Ellis, C., K. Keddara, G. Rozenberg: *Dynamic Change Within Workflow Systems.* In Proc. Conference on Organizational Computing Systems (COOCS) 1995, 10–22

[3] Georgakopoulos, D., M. Hornick, A. Sheth: *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure*. Distributed and Parallel Databases, 3:119–153, 1995

[4] Hagemeyer, J., T. Herrmann, K. Just-Hahn, R. Striemer: *Flexibility in Workflow Management Systems (in German)*. Software-Ergonomie '97, 179–190, Dresden, March 1997.

[5] Han, Y., Sheth, A., Bussler, C.: *A Taxonomy of Adaptive Workflow Management.* In Proceedings CSCW-98 Workshop: Towards Adaptive Workflow Systems. (downloaded from http://ccs.mit.edu/klein/cscw98/paper03 on 09-24-1998)

[6] Joeris, G.: *Aspects and Concepts of Flexibility in Workflow Management Systems.* (in German) In Proc. D-CSCW98 Workshop on Flexibility and Cooperation in Workflow Management Systems, Dortmund, Sept 1998. Technical Report Angewandte Mathematik und Informatik 18/98-I, University of Muenster, Germany 1998

[7] Klein, M. (Ed.): *Towards Adaptive Workflow Systems.* Workshop in The 1998 ACM Conference on Computer Supported Cooperative Work, Seattle, Washington, November 14-18, 1998 (Online Proceedings at http://ccs.mit.edu/klein/cscw98/ on 09-24-1998)

[8] Leymann, F., W. Altenhuber: *Managing Business Processes as an Information Resource*. IBM Systems Journal 33, 1994, 326–347

[9] Luo, Z., Sheth, A.: *Defeasible Workflow, its Computation and Exception Handling.* In Proceedings CSCW-98 Workshop: Towards Adaptive Workflow Systems. (downloaded from http://ccs.mit.edu/klein/cscw98/paper10 on 09-24-1998)

[10] Reichert, M., P. Dadam: *Supporting Dynamic Changes of Workflows Without Loosing Control.* Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management, Vol. 10, No. 2, 1998

[11] Sheth, A., D. Georgakopoulos, S.M.M. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, A. Wolf: *Report from the NSF Workshop on Workflow and Process Automation in Information Systems.* Technical Report UGA-CS-TR-96-003 University of Georgia, Athens, GA, 1996

[12] Siebert, R., Weske, M. (Eds.): *Flexibility and Cooperation in Workflow Management Systems.* (in German) Workshop at D-CSCW98, German Conference on CSCW 1998, Dortmund, Sept 1998. Technical Report Angewandte Mathematik und Informatik 18/98-I, University of Muenster, Germany 1998

[13] Weske, M., Vossen, G.: *Workflow Languages.* In: P. Bernus, K. Mertins, G. Schmidt (Editors): Handbook on Architectures of Information Systems. (International Handbooks on Information Systems), pp 359–379. Berlin: Springer 1998

[14] Weske, M., Hündling, J., Kuropka, D., Schuschel, H.: *Object-Oriented Design of a Flexible Workflow Management System.* (in German) Informatik Forschung und Entwicklung. 13(4) 1998, pp 179–195. Berlin: Springer 1998

[15] Weske, M.: *Design and Implementation of an Object-Oriented Workflow Management System.* Fachberichte Angewandte Mathematik und Informatik 33/98-I, Universität Münster 1998

Defining Flexible Workflow Execution Behaviors

Gregor Joeris

Intelligent Systems Department, TZI - Center for Computing Technologies
University of Bremen, PO Box 330 440, D-28334 Bremen
joeris@informatik.uni-bremen.de

Abstract. *In this paper, we focus on the definition and adaptation of the execution behavior of a task in order to support flexible workflows in the presence of distributed workflow enactment. We argue that an adequate behavior definition is the basis for both, modeling less-restrictive workflows in advance as well as supporting dynamic workflow changes. We show how different control flow dependency types can be specified in our approach and can be used to define less-restrictive workflows. Furthermore, we discuss the definition of an adequate behavior for dynamic modifications in different situations. In particular, we describe how the application of and reaction to dynamic changes can be adapted in our approach depending on the process context and the behavior of a task itself.*

1. Introduction

The development of process-model based workflow management systems (WFMS) has been driven mostly by focussing well-structured business processes from the viewpoint of transactional processing. WFMSs are applied following a rigorous methodology of business process (re-)engineering and formal workflow specification which leads to well-defined processes and is well-suited for production workflows. This restricted view – which is not inherently caused by the workflow paradigm – is the main reason for the inflexibility of today's WFMS and their underlying process representation formalisms. Support for flexible workflows in process-model based WFMS has to cope with two fundamental challenges:

(a) *A-priori flexibility* focus on the specification of a flexible workflow execution behavior to express an accurate and less restrictive behavior in advance; flexible and adaptable control and data flow mechanisms have to be taken into account in order to support ad hoc and cooperative work at the workflow level (cf. [ElNu96]).

(b) *A-posteriori flexibility* (flexibility by dynamic adaptation) is provided by the change and evolution of workflow models in order to modify workflow specifications on the schema and instance level due to dynamically changing situations of a real process (cf. [EKR95, CCPP96, ReDa98, JoHe98]). Note, that in the case of dynamic modifications we also have to define a-priori when, i.e. in which context and in which state of execution, certain modifications are allowed in order to ensure the dynamic and semantical consistency of a process.

Thus, the definition of the behavior of workflow execution as well as workflow evolution are the basis for supporting flexible workflows. In this paper, we focus on the workflow behavior definition and adaptation in both cases in the presence of a distributed workflow enactment approach which forms the basis for a scalable workflow management system. First, we sketch our workflow modeling language and show how the behavior of a task can be defined and adapted in different contexts. In particular, user-defined control flow dependencies, which allow to define and reuse complex behavior patterns, can be specified and applied in different contexts. Next, we outline how these concepts can be used to define a-priori less-restrictive workflows, i.e. workflows where a certain degree of freedom is left open to the actor. Furthermore, we discuss the definition of an adequate behavior for dynamic modifications in different situations without restricting our self on a transactional view on processes. In particular, we show how the application of and reaction to dynamic changes can be adapted in our approach depending on the process context and the behavior of a task itself.

2. The Workflow Modeling Approach

2.1 Task and Process definition

The building block of our workflow modeling approach is a *task definition* (or task type) which consists of a *task interface,* that specifies 'what is to do', and potentially several *process definitions,* which specify how the task may be accomplished. The task interface is defined by attribute, parameter, and process constraint defini-

tions (all neglected throughout this paper) and by a task behavior definition which specifies the external context-free behavior of a task (e.g., transactional or non-transactional). The context-dependent behavior of task is defined by its application within a process definition. A process definition[1] may be atomic consisting only of a task description or system invocation, or complex. A complex process is defined in an activity-oriented manner by a task graph. The decision is taken at run-time, which process definition of a task definition is used to perform a task (late binding). This late binding mechanism also allows to create a new process definition at run-time (late modeling).

A *task graph* consists of task components, connectors, start and end nodes, and data inlets and outlets, which are linked by control and data flow dependencies: A *task component* is an applied occurrence of a task definition within a process definition. For every task component a split and join type (AND / OR) can be specified. Furthermore, connector components are predefined which just realize splits and joins.

Task components (and start and end node) are linked by *control flow dependencies*. Iterations within this task graph are modeled by a special predefined feedback relationship. A condition can be associated to every dependency to support conditional branches. We allow to define different control flow dependency types which can be applied and reused within several process definitions. The semantics of a control flow dependency type is defined by ECA rules as shown in the next section and illustrated in figure 2.

Similar to the definition of control flow dependencies we support the definition of *group relationship types*. A group relationship is used within a process definition in order to group arbitrary task components of a task graph; it applies the behavior defined by the group relationship to its components. A task component can be part of several not necessarily nested groups. A special kind of a group is a block. Blocks are nested and contain a subtask-graph with exactly one start and end component (particularly useful for exception handling).

Finally, task components can be linked by *dataflow relationships* according to the input and output parameters of their task definitions. Furthermore, a data inlet (or outlet) is used as a data source (or sink) in order to realize a vertical dataflow between the parameters of the task definition and their use within the workflow.

2.2 Distributed Workflow Enactment and its Execution Semantics

The execution semantics of tasks and of the task graph is defined by a statechart variant and event-condition-action (ECA) rules. Our enactment model is based on treating tasks as *reactive components* which encapsulates their internal behavior and interact with other tasks by message/event passing. This is a natural basis for a distributed enactment of workflows which was one of the design goals of our approach (beside of flexibility), and it is essential for scaling up to enterprise-wide workflow support.

A task has several built-in operations, which can be categorized into state transition operations, actor assignment operations, operations for handling of (versioned) inputs and outputs, and workflow change operations. For every operation, the task has the knowledge about when to trigger the operation, a condition that must hold for executing the transition and that acts as guard, and a list of receivers to which events are passed (to avoid communication overhead, we use no broadcast).

Before we introduce the behavior definition and adaptation on schema level, we briefly sketch syntax and semantics of our ECA rules (see [JoHe99a] for details; examples are given in figure 1 and 2): First of all, an ECA rule is always associated with an operation/transition, which defines the action part of the rule. Thus, ECA rules are structured according to the task's transitions. Additionally, an ECA rule consists of a list of event captures, a condition, and a receiver expression. Events define when an operation is to be triggered: when a task receives an event that matches an event capture in the event capture list, and when the task is in the source state of the corresponding transition, the event is consumed and the task tries to perform the transition. The invocation of a transition causes the evaluation of its condition. This transition condition acts as a guard, i.e., the transition is performed only when the condition holds (otherwise nothing is done). Thus, we follow a state-based semantics where a transition can be triggered by (internal and atomic) events or externally by user invocation. Invocation and applicability of a transition are strictly separated. The matching of an event with an event capture can be qualified to the causing task. For example, this allows a task to react differently on the event 'fin-

---

[1] Note, that we use process definition in a more restricted sense defining only how a task has to be done. To avoid misunderstandings, we use workflow and workflow specification as a general term (independent from our approach) in the usual more broader sense.

ished' depending on whether the event was received from a predecessor or from a sub-task. Furthermore, a trigger condition can be specified within an event capture which must hold for a valid event capture. Otherwise, the next event capture which matches the event is searched.

2.3 Definition and Adaptation of the Execution Behavior

The *context-free* behavior of a task is given by the behavior definition of a task definition by means of a statechart variant. It defines the states, their decomposition, and the operations/transitions which can be invoked in a certain state. Furthermore, exactly one context-free ECA rule can be defined for every transition. A task definition can *inherit* from an abstract task definition, i.e., a task definition with no process definitions, in order to define behavior classes of tasks (e.g., non-transactional, transactional; cf. [KrSh95, Wes98]). Within an inherited statechart, states can be refined and transitions can be added and redefined by changing their associated ECA rule. This allows to adapt the context-free behavior of tasks. Every task definition inherits from a *predefined task definition,* which consists of a statechart that defines the basic states, transitions, and context-free ECA rules as illustrated in figure 1.
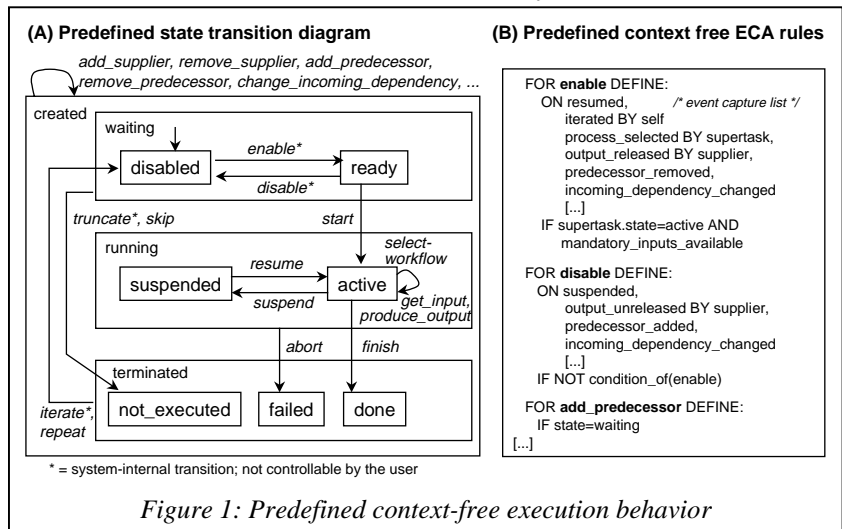

Figure 1: Predefined context-free execution behavior

The *context-dependent* behavior is given by the control flow dependencies and groups within a process definition. Rather than providing a limited set of different built-in control flow dependency types and group relationship types, arbitrary control flow types can be defined and adapted by a process engineer (cf. [JaBu96]). They are defined by a label, an informal description, and a set of ECA rules, which give the semantics of the dependency type. Within the task graph, the control flow dependencies or group relationships can be used by their labels abstracting from the detailed definition and reusing complex behavior patterns. Control flow dependency types are used to define ECA rules which establish intertask dependencies (e.g. end-start, start-start, deadline). Group relationship types are used to apply a behavior pattern to an arbitrary set of components.

As an *example*, we briefly explain the definition of the standard end-start dependency which consists of several rules partially shown in figure 2. We concentrate on the first rule which is defined for the enable transition and is applied to the target component of the dependency. The event capture defines that the enable transition is triggered whenever an event 'finished' has been received along the standard dependency under the condition that the corresponding dependency condition (specified by the placeholder 'dependency_condition') evaluates to true. The condition of the transition is defined by an reference to the source component requiring that it is an state done. The receiver expression is omitted in all examples.

In order to obtain the behavior of a task instance, the (partially) defined ECA rules of the context-free behavior are joined with the ECA rules that the task instance inherits from its context, i.e. from its dependencies and group relationships within a process definition. An example is shown in figure 2 for the 'FunctionalCheck' task which obtains the behavior of the incoming standard end-start dependency from 'Design', of the group type 'Exclusion', and of the context-free statechart. There are different modes for merging the context dependent behavior definitions. These modes are defined for every ECA rule (omitted in Fig. 2) and are applied to the transition condition: (a) conjunctive (default for group relationships) (b) disjunctive (c) using the join type of the component (default for dependencies) (d) using the inverted join type of the component (e) overriding. Dependency or group types using the overriding mode for the same transition cannot be applied simultaneously to a component. The overriding mode is normally used for adapting the application condition of change operations (see section 3.3).
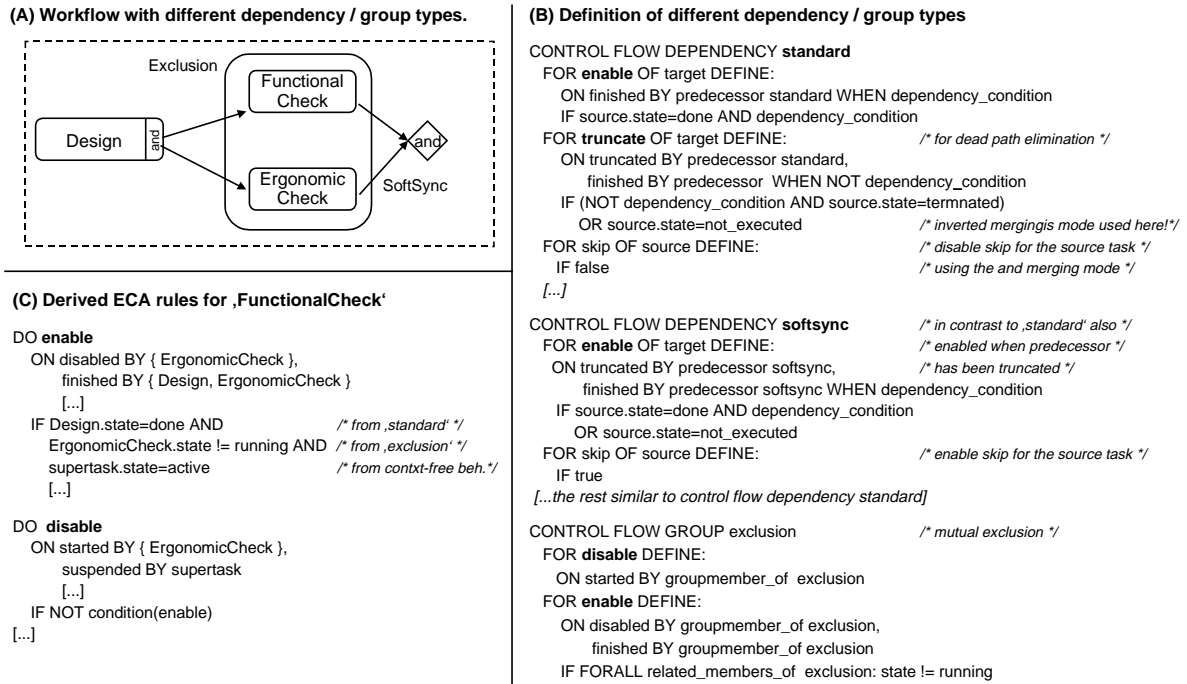
(A) Workflow with different dependency / group types.

```
                    Exclusion
                  ┌──────────────┐
                  │  Functional  │
        ┌───────┐ │    Check     │
        │       ┌┤ └──────────────┘┐
        │ Design│ └                 \◇
        │       └┤ ┌──────────────┐ /and
        └───────┘a│  Ergonomic   ├─┘
               n d│    Check     │  SoftSync
                  └──────────────┘
```

(B) Definition of different dependency / group types

```
CONTROL FLOW DEPENDENCY standard
  FOR enable OF target DEFINE:
    ON finished BY predecessor standard WHEN dependency_condition
    IF source.state=done AND dependency_condition
  FOR truncate OF target DEFINE:                    /* for dead path elimination */
    ON truncated BY predecessor standard,
        finished BY predecessor  WHEN NOT dependency_condition
    IF (NOT dependency_condition AND source.state=termnated)
      OR source.state=not_executed                  /* inverted mergingis mode used here!*/
  FOR skip OF source DEFINE:                         /* disable skip for the source task */
    IF false                                         /* using the and merging mode */
  [...]

CONTROL FLOW DEPENDENCY softsync          /* in contrast to ‚standard' also */
  FOR enable OF target DEFINE:             /* enabled when predecessor */
    ON truncated BY predecessor softsync,  /* has been truncated */
        finished BY predecessor softsync WHEN dependency_condition
    IF source.state=done AND dependency_condition
      OR source.state=not_executed
  FOR skip OF source DEFINE:               /* enable skip for the source task */
    IF true
  [...the rest similar to control flow dependency standard]

CONTROL FLOW GROUP exclusion              /* mutual exclusion */
  FOR disable DEFINE:
    ON started BY groupmember_of  exclusion
  FOR enable DEFINE:
    ON disabled BY groupmember_of exclusion,
        finished BY groupmember_of exclusion
    IF FORALL related_members_of  exclusion: state != running
```

(C) Derived ECA rules for ‚FunctionalCheck'

```
DO enable
  ON disabled BY { ErgonomicCheck },
      finished BY { Design, ErgonomicCheck }
      [...]
  IF Design.state=done AND              /* from ‚standard' */
    ErgonomicCheck.state != running AND /* from ‚exclusion' */
    supertask.state=active              /* from contxt-free beh.*/
    [...]

DO disable
  ON started BY { ErgonomicCheck },
      suspended BY supertask
      [...]
  IF NOT condition(enable)
[...]
```

Figure 2: Different control flow types and their application for the definition of flexible workflows

3. Behavior Definition and Adaptation for Flexible Processes

In this section, we show how the introduced concepts of behavior definition can be used to define a priori flexible workflows and to adjust the application condition of and the reaction to dynamic changes depending on the behavior of the task itself and the context.

3.1 Defining Less-restrictive Workflows

We give two examples of defining a-priori less-restrictive workflows, i.e. workflows where a certain degree of freedom is left open to the actor, using the introduced concepts of user-defined control flow types.

The first example is the definition of the group relationship type 'exclusion' which is shown in figure 2. It forces its members to execute mutually exclusive. In conjunction with parallel branches, we can define that certain tasks should be executed sequentially, but without defining the actual ordering of the tasks. So, the user can choose which task he or she wants to perform next. In our example, a QA engineer may decide whether he/she wants to check a specification first against the functional or against the ergonomic requirements.

The second example focuses on skipping a task. Since skipping a task is normally not desirable (except from the viewpoint of the actor) and probably cause serve problems when needed output data has not been produced, its application should be restricted. Furthermore, in our state-based semantic, skipping a task would result in a deadlock since the successor tasks would wait for termination of the skipped task. Both problems are solved by the control flow dependency type 'softsync' (cf. [ReDa98], which also show useful applications in the case of dynamic changes). It waits only for the termination of a task when the task still can be executed (see relaxed enable condition in figure 2). Furthermore, the dependency allows to control when skipping is enabled. The skip transition is disabled for tasks which have at least one outgoing standard dependency. Thus, the process modeler can define which tasks can be skipped in a certain workflow.

3.2 Supporting Collaborative Workflows

Collaborative workflows are supported in our approach by version and workspace control capabilities which are integrated with the workflow model on conceptual level. Consumed and produced versions are managed within a task-oriented workspace. Versions can be released for dedicated tasks which allows a versioned data
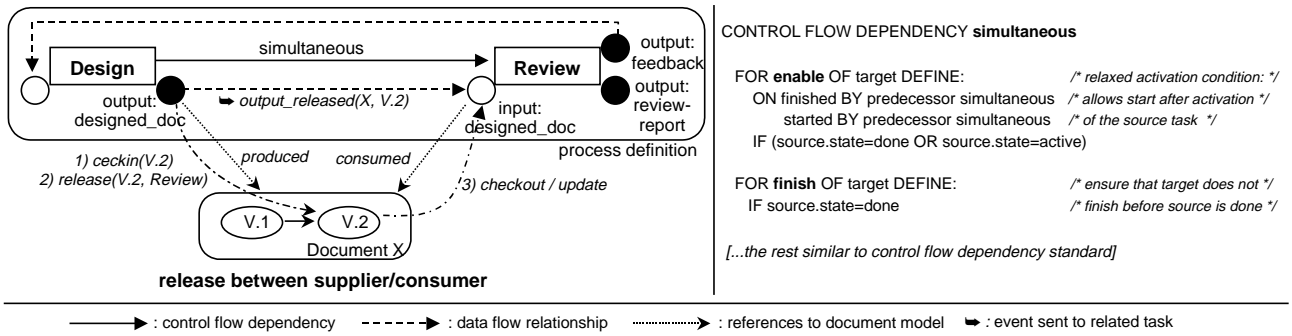
Figure 3: Data exchange between running tasks – controlled cooperation on workflow level

flow and the exchange of intermediate results between tasks. Furthermore, the data flow can be also used for control flow purposes. The availability of input data can be checked and the operations for releasing outputs generate events as any other transition so that tasks can react on these events. For example, the event 'output_released' triggers the evaluation of the enable transition of the consumers (see figure 1 and 3).

It is out of the scope of this paper to present details about the workspace capabilities (see [Joe98]). Rather, we like to show, how data exchange between simultaneously active instances can be controlled on the workflow level by means of the simultaneous dependency (cf. [HJKW96]). The definition of this dependency is illustrated in figure 3. It relaxes the activation condition so that the dependent task does not have to wait for the termination of the preceding task; a task is enabled when all mandatory inputs are available and the preceding task has been started. The main purpose of the simultaneous dependency is to ensure that the dependent task does not terminate before the preceding task. This termination synchronization guarantees that the latest results of the supplier is processed by the consumer. An example application is the design and review of a technical document where the designer may request for an early feedback from the reviewer (see figure 3).

3.3 Situation-dependent Handling of On-the-fly Changes

Our approach to dynamic changes of enacting workflow instances is based on applying ECA rules also to change operations. Every change primitive is encapsulated by a pre-condition which restricts its application, and by raising a corresponding event which is handled by the affected instances in order to ensure the behavioral consistency of the execution states. Thus, conceptually a change operation can be treated like a state transition, and on-the-fly changes are supported in the presence of distributed workflow enactment since every task instance object has the knowledge about how to react on a change. The basic idea of this approach has been presented in our previous work DYNAMITE [HJKW96] on software process management, and all details of our approach to managing evolving workflow specifications can be found in [JoHe99b].

Whether a change is allowed and how to react on it highly depends on the particular situation and the behavior of the involved tasks. In this paper, we concentrate on the situation-dependent handling of dynamic changes. As an example, we outline useful application conditions and reactions for adding a new predecessor task (as an insertion between two sequential tasks or as a new parallel branch). In this case, the insertion of a new control flow dependency is important. It affects the target component which depends on a new predecessor task. Depending on the behavior of the affected task and on the context in which the task is applied, different application conditions and reactions can be useful for this change which all can be realized in our approach:

1) In general, this modification can be allowed for target components which have *not yet been started*. When the dependent task is already in the state ready, the event 'predecessor_added', which is raised by the change operation, results in triggering the disable transition. This transition is performed only if the enable condition no longer holds (see figure 1). Thus, re-evaluation of the enable condition ensures the behavioral consistency. However, adding a new predecessor task may be also useful and can be handled for target components which are already active or have been finished. In contrast to [EKR95], [CCPP96], [ReDa98], or [HoJa98], we do not restrict ourselves to situation-independent theoretical correctness criteria:

2) If the dependent task is already *active*, a meaningful reaction for example for an automatic batch process is to abort the task and to restart it later on (probably performing additional compensation activities). Fur-

Extended Statechart for tasks with compensation
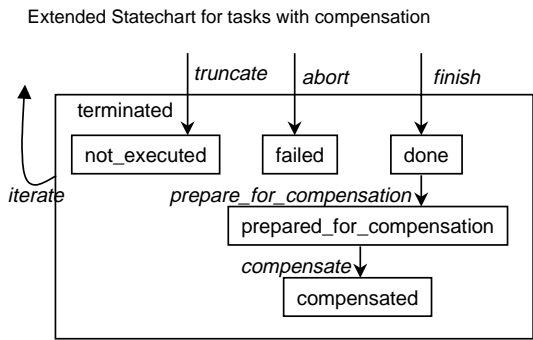
CONTEXT-FREE ECA rules
  FOR **prepare_for_compensation** DEFINE: /* changes to finished task triggers */
    ON predecessor_added, ...   /* preparation for compensation */
  FOR **compensate** DEFINE: /* compensation starts when all */
    ON prepared_for_compensation BY self,   /* successors have been compensated */
    IF FORALL sucessors compensational: state = compensated

CONTROL FLOW DEPENDENCY **compensational**
  FOR **disable** OF target DEFINE: /* disable ready tasks when predecessor */
    ON prepared_for_compensation BY predecessor compensational   /* will be compensated */
  FOR **abort** OF target DEFINE: /* and abort running tasks in this case */
    ON prepared_for_compensation BY predecessor compensational
  FOR **prepare_for_compensation** OF target DEFINE: /* transitively prepare for compensation */
    ON prepared_for_compensation BY predecessor compensational,
  FOR **compensate** OF source DEFINE: /* trigger compensation in reverse order */
    ON compensated BY successor compensational

*Figure 4: Behavior definition for tasks with compensation*

thermore, a manual task may just be suspended and can be resumed when the new preceding task is done. A human actor can easily work on the changed input values/documents.

Both behaviors can be realized by defining different behavior classes for batch tasks and manual tasks. In the former case, we add a trigger 'ON predecessor_added' to the context-free ECA rule of the abort transition, in the latter case, we add the same trigger to the suspend transition. Furthermore, we relax the transition condition of add_predecessor to 'IF state=waiting OR state=running'.

3) If the target task has been already *finished*, we may compensate all succeeding task of the new task (if possible). In figure 4, the behavior of tasks which provide compensation facilities is shown. In our example, compensation is done with a two phase protocol which takes the ordering of the tasks' execution into account (cf. [Ley95, KaRa98]): First, all tasks which are compensational dependent are prepared for compensation. Next, these tasks are compensated in the opposite ordering of their former execution. This complex behavior is realized by the compensation dependency. A compensation which is not order-dependent can be realized easily by a group type which just define an ECA rule that triggers the compensation when a certain event occurs.

However, assume that the affected tasks are part of an iteration loop and that we are interested only in the future execution; then, we can insert the new task without any impact on the current pass of the loop (in the sense "now it's too late, but next time we should perform the additional task"). The new task becomes relevant only for the next iteration. This is realized by a group relationship type 'allow_post_execution_change' (shown in figure 5) which is used in a process definition to mark those regions where the above mentioned situation should be supported (obviously, this policy is not useful in general). Note, that the interplay of context-free and context-dependent behavior definition results in the appropriate behavior for all mentioned situation. E.g., for a manual task changes may be allowed in any state whereas for a automatic task changes may be disallowed when the task is active (if rollback/compensation is not possible or desirable).

4) Finally, for some parts of a process a process engineer may want to disallow dynamic modifications. In this case, a group relationship type 'disallow_change', which disables all change operations using the overriding mode, can be used to mark the relevant parts of the process (cf. figure 5).
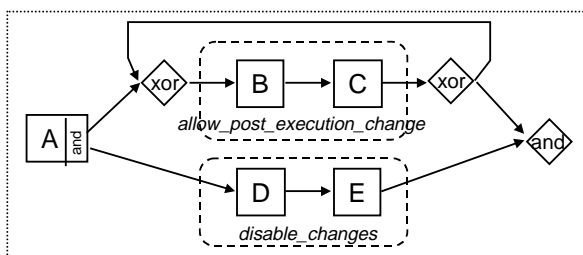
CONTROL FLOW GROUP disable_changes
  FOR **add_predecessor** DEFINE: /* same definition for all */
    override: IF false   /* change operations */
  [...]

CONTROL FLOW GROUP **allow_post_execution_change**
  FOR **add_predecessor** DEFINE
    IF state=terminated

Figure 5:Situation-dependent adaptation of the behavior of dynamic changes

4. Conclusion

The definition and situation-dependent adaptation of the tasks execution behavior is the basis for both, providing a priori flexible workflows and supporting dynamic changes in every possible situation. For the latter case, it is essential to note that there exist several practical cases where theoretical correctness properties, in particular the compliant property (cf. [CCPP96]), are too restrictive. We have shown, that the definition of control flow dependency and group relationship types on the basis of ECA rules is a powerful concept for behavior adaptation and dynamic changes in the present of distributed workflow enactment. Finally, human actors can react very flexible on changes of the context of a task. The integration of version and workspace control capabilities substantially supports adequate reactions to workflow changes in the case of manual and cooperative tasks.

The workflows which can be defined in this approach are by far more complex than in the case of workflows which consists only of conditional and parallel branches. Therefore, the analysis of correctness properties (e.g. deadlock-freeness) of the resulting task behaviors and their interaction are is a hard problem on which we currently work. The introduced concepts have been implemented in the project MOKASSIN which is funded by the German Ministry for Research and Technology (BMBF). Experiences as well as the architecture of our system which is realized as an distributed object system based on CORBA will be discussed in subsequent papers.

References

[CCPP96] Casati F., Ceri, S., Pernici B., Pozzi, G.: „Workflow Evolution". In *Proc. of 15th Int. Conf. on Conceptual Modeling (ER'96)*, Cottbus, Germany, 1996; pp. 438-455.

[ElNu96] Ellis C.A. and Nutt, G.J.: "Workflow: The Process Spectrum", in *NSF Workshop on Workflow and Process Automation in Information Systems*, Athens, Georgia, 1996.

[EKR95] Ellis C.A., Keddara K. and Rozenberg, G.: "Dynamic Change Within Workflow Systems", in *Proc. of the Int. Conf. on Organizational Computing Systems COOCS'95*, Milpitas, CA, 1995; pp. 10-21.

[HJKW96] Heimann, P.; Joeris, G.; Krapp, C.-A.; Westfechtel, B.: "DYNAMITE: Dynamic Task Nets for Software Process Management", in *Proc. of the 18th Int. Conf. on Software Engineering,* Berlin, Germany, 1996; pp. 331-341.

[HoJa98] Horn S. and Jablonski S.: "An Approach to Dynamic Instance Adaption in Workflow Management Applications", in *Proc. of the CSCW-98 Workshop - Towards Adaptive Workflow Systems*, Seattle, WA, Nov. 1998

[JaBu96] Jablonski, St.; Bussler, Ch.: "Workflow Management - Modeling Concepts, Architecture and Implementation", International Thomson Computer Press, London, 1996.

[Joe98] Joeris, G.: "Aspekte und Konzepte der Flexibilität in Workflow-Management-Systemen", *in D-CSCW'98 Workshop on 'Flexibilität und Kooperation in Workflow-Management-Systemen'*, Technical Report Angewandte Mathematik und Informatik 18/98-I, University of Münster, 1998; pp. 3-12.

[JoHe98] Joeris, G; Herzog, O.: "Managing Evolving Workflow Specifications", in *Proc. of the 3rd Int. IFCIS Conf. on Cooperative Information Systems (CoopIS'98)*, New York, Aug. 1998, IEEE Computer Society Press; pp. 310-319.

[JoHe99a] Joeris G.; Herzog O.: "Towards Flexible and High-Level Modeling and Enacting of Processes", in *Proc. of the 11th Int. Conf. on Advanced Information Systems Engineering (CAiSE'99)*, LNCS 1626, Springer-Verlag, 1999; pp. 88-102.

[JoHe99b] Joeris G.; Herzog O.: " Managing Evolving Workflow Specifications With Schema Versioning and Migration Rules", TZI Technical Report 15-1999, Center for Computing Technologies (TZI), University of Bremen, 1999.

[KaRa98] Kamath, M.; Ramamrithan, K.: "Failure Handling and Coordinated Execution of Concurrent Workflows", in *Proc. of the 14th Intl. Conf. on Data Engineering (ICDE'98)*, Orlando, Florida, Feb. '98.

[KrSh95] Krishnakumar, N.; Sheth, A.: "Managing Hetereogeneous Multi-system Tasks to Support Enterprise-wide Operations", in *Distributed and Parallel Databases*, 3, 1995; pp. 1-33.

[Ley95] Leymann, F.: "Supporting Business Transactions Via Partial Backward Recovery in Workflow Management Systems, in Lausen, G. (Hrsg.): *Datenbanksysteme in Büro, Technik und Wissenschaft. GI-Fachtagung*, Springer Verlag, 1995; pp. 51-70.

[ReDa98] Reichert, M; Dadam, P.: "ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control", *Journal of Intelligent Information Systems - Special Issue on Workflow Managament*, 10(2), 1998; pp. 93-129.

[Wes98] Weske, M: "State-based Modeling of Flexible Workflow Executions in Distributed Environments", in Ozsu, T.; Dogac, A.; Ulusoy, O. (eds.) *Proc. of the 3rd Biennial World Conference on Integrated Design and Process Technology (IDPT'98), Volume 2 – Issues and Applications of Database Technology*, 1998; pp. 94-101.

Enterprise-Wide and Cross-Enterprise Workflow Management: Challenges and Research Issues for Adaptive Workflows

Manfred Reichert, Thomas Bauer, Peter Dadam

Dept. Databases and Information Systems, University of Ulm
{reichert, bauer, dadam}@informatik.uni-ulm.de
http://www.informatik.uni-ulm.de/dbis

Abstract

The paper discusses important challenges and research issues for adaptive workflow management systems (WfMS), especially if they shall be applied to enterprise-wide applications. It shows that an adaptive WfMS must provide support for different kinds of (dynamic) workflow (WF) changes in order to be applicable to a broader spectrum of processes. In this context, both, requirements for WF schema evolution and issues related to ad-hoc changes of individual WF instances are discussed. A particulary interesting aspect, which is described in more detail, is how to combine such dynamic changes with a distributed execution of workflows, taking into account performance issues. For enterprise-wide and cross-enterprise workflows, the distributed execution of workflows may be attractive due to several reasons.

1 Introduction

Workflow Management Systems (WfMS) offer a promising technology that has the potential to change the implementation of enterprise-wide and cross-enterprise application systems significantly. In fact, only this technology makes it possible to realize process-oriented application systems in larger quantities and at affordable costs. To adequately support enterprise-wide and cross-enterprise applications, a WfMS must not only cope with a large number of users and concurrently active workflow (WF) instances, but it must also cover a broad spectrum of processes. In this context, high flexibility, maintainability, and scalability are essential requirements. In the ADEPT[1] project we, therefore, have spent a lot of effort especially on these subjects [BaDa97, BaDa98, BaDa99a, DaRe98, ReDa98, RHD98].

In this paper we discuss important challenges and research issues for adaptive WfMS [CCPP98, JoHe98, RHD98, Sieb98, Wesk98], especially if they shall be applied to enterprise-wide and cross-enterprise processes. We denote a WfMS as *adaptive* if it supports run-time changes of in-progress WF instances. Such adaptations become necessary, for example, when new tasks have to be added to a WF instance at run-time or when pre-defined control or data flow dependencies between WF activities have to be dynamically changed [ReDa98]. In such cases the execution of the WF instance must be (partially) suspended, the modification of its WF instance graph, its attributes and/or its state be performed, and afterwards its execution be resumed. The paper is organized as follows: In Section 2 we show that an adaptive WfMS must support different kinds of (dynamic) WF changes in order to be applicable to a broader spectrum of processes. We discuss major requirements for the evolution of WF schemes and for the concomitant run-time adaptation of corresponding WF instances. Furthermore we deal with issues related to ad-hoc changes of individual WF instances and we discuss problems that arise from the integrated support of both kinds of changes. A particularly interesting aspect is discussed in Section 3, namely how to combine dynamic WF changes with a distributed execution of workflows, taking into account performance issues. For enterprise-wide and cross-enterprise workflows, the distributed execution of workflows may be advantageous in several respects. In Section 4 we summarize further issues. The paper concludes with a summary and an outlook on future work.

---

[1] ADEPT stands for <u>A</u>pplication <u>D</u>evelopment based on <u>E</u>ncapsulated <u>P</u>re-Modeled Process <u>T</u>emplates.

2   Workflow Schema Evolution and Support of Ad-hoc Deviations

Enterprise-wide and cross-enterprise business processes may change rather frequently [RHD98]. Once an application system has been made to behave strictly process-oriented, it must be adjustable to process changes and to evolving organizational structures very quickly and at reasonable costs. Such adaptations may affect WF templates or other aspects of the process-oriented application system (e.g., the model capturing organizational entities). In any case, they must be performed without causing inconsistencies between the different models of the process-oriented application (e.g., faulty references).

In order to increase the robustness of the WF-based application system, it is very important to detect and to eliminate design and implementation errors as early as possible. Potential problems introduced by the concept of process-orientation are that the process may block itself during execution, may never be able to enter certain branches of the WF graph, may not meet temporal constraints (e.g., minimal or maximal time distances between the execution of two activities), or may invoke activity programs with missing or incomplete input data, for example. To avoid such problems, an adequate formal basis must be provided for WF modeling. On the one hand such a formal model must allow WF designers to capture business processes as naturally as possible and in a way understandable to the user. On the other hand, it must enable formal verifications for the absence of deadlocks, the proper termination of the flow, the correctness of the data flow, or the consistency of a time schedule, to name a few examples. With respect to consistency issues, commercial WfMS show severe limitations that disqualify them for the support of sophisticated WF applications. – In the ADEPT project we have exhaustively investigated WF modeling issues and we have developed the ADEPT$_{base}$ model as a formal basis (for details see [ReDa98]).

The use of a formal WF model and the provision of corresponding analysis algorithms are also prerequisites for keeping maintenance costs low. Structural modifications of a WF schema (*WF type changes*), like the insertion, deletion, or shift of process steps, must not lead to undesired side-effects and program failures. Instead, the system must assist the WF designer in detecting all concomitant schema modifications, which become necessary in order to guarantee a robust and correct execution for (new) WF instances of the changed WF type. To maintain the correctness of a WF schema, however, is only one of several requirements for *WF schema evolution*. Changing a WF type does also mean, in general, that we still have WF instances active in the system that follow the "old" schema. Especially for long-running processes, it is desirable to automatically adapt them to the new process structure as well (as far as this is possible). As the WF instances may be in different states, however, the respective schema modifications may be propagated only to a subset of them [CCPP98, JoHe98]. For example, an already completed activity must not be deleted from a WF instance graph. For the same reason, a new activity cannot be inserted into a region of a WF instance graph, which has already been processed. For modifications of a loop body, another important aspect has to be considered: A change, which is not valid in the current state of a loop iteration, may become applicable when the loop enters its next iteration. In such a situation, it is favorable to record the change and to apply it when this loop back will be performed. In any case, the WfMS must ensure a correct and stable execution behavior afterwards. Again, the provision of a formal model can help a lot, since it will allow the system to check whether a particular change is valid in the current state of a WF instance or whether it is not. In the former case, the WfMS must also adapt the state of the WF instance after its modification [ReDa98]. Since there may be a large number of active instances of the same WF type in the system, the necessary checks and adaptations must be performed efficiently and automatically by the WfMS. Finally, the WfMS must cope with the co-existence of WF instances following either the old or the new schema. This presumes appropriate concepts and mechanisms for the versioning of WF schemes and for the adequate representation of WF instances [JoHe98].

Things become even more complicated if *ad-hoc deviations* from the pre-defined WF schema must be supported at the instance level. Examples for such run-time deviations are the deletion of one or more WF steps (e.g., to skip their execution) from a WF instance graph or the dynamic insertion of a new activity. Such interventions into the control of a WF instance may become necessary to handle *exceptional situations* [MüRa99, RHD98] or to model parts of the WF that cannot be completely pre-defined at build-time (*late modeling*). In our experience, due to combinatorial reasons, for more complex processes it is neither possible nor cost-effective to capture all possible task sequences and all exceptions in advance. But even for simple workflows, unpredictable situations may occur that require ad-hoc deviations from the pre-planned process at run-time [RHD98]. Such ad-hoc deviations must not lead to consistency problems or to an unstable system behavior (e.g., program failures due to the invocation of an activity with missing input data). This means, in fact, that one has to show that none of the correctness guarantees or assertions,

which have been achieved by formal checks at build-time, are being violated by the introduction of ad-hoc changes at run-time. Instead the system must assist the user in detecting all concomitant adaptations that are necessary to ensure a robust and correct execution behavior afterwards. To achieve this, all aspects of the WF model (control flow, data flow, temporal constraints, etc.) and their possible interactions have to be taken into consideration. For example, when deleting a step from a WF instance graph, the data flow may have to be adapted as well in order to preserve its correctness. Due to the numerous interdependencies that exist between the different aspects of a WF model, with increasing expressiveness of the used WF meta model, it becomes more and more difficult to handle ad-hoc changes in a correct and consistent manner. Therefore, it is extremely important to hide details of a change (like e.g., the complexity arising from the re-mapping of input and output parameters of WF activities) from the user. Again, an adequate formal model can help a lot. Such a model must offer a clear semantics, which enables the system to argue on correctness issues and which covers all possible cases, either by supporting the desired action or by rejecting it (no implementation holes). Ideally, such a model enables the system to restrict the necessary analysis to a portion of the WF instance graph in most cases and, by doing so, to perform the necessary checks very efficiently. Further details and a discussion of other issues related to dynamic WF changes can be found in [ReDa98, RHD98].

Generally, an adaptive WfMS has to consider both kinds of changes, i.e., it must support adaptations of a potentially large number of WF instances to modifications of their WF type as well as ad-hoc changes of single WF instances. To adjust in-progress WF instances to a type change, however, is a non-trivial problem if ad-hoc changes have to be supported as well. In this context, the main problem arises from the fact that the instances of a WF type may not only be in different states when a type change shall be propagated, but may also have a process structure (represented by the WF instance graph) that deviates from that of their original schema. While in some cases this may not affect the applicability of the type change to a particular instance, in other cases there may exist unresolvable conflicts between previously applied ad-hoc changes of a WF instance graph and the type change. Well, how can WF type changes be efficiently propagated to a potentially large number of WF instances under these conditions? In principle, for each instance of a modified WF type, the system must check whether the type change is currently applicable to the corresponding WF instance graph or not. As a large number of WF instances may have to be adapted to the type change, the necessary checks should be automatically performed by the WfMS without causing a performance penalty. Costly user interactions, which are also not tolerable due to robustness and correctness reasons, must be avoided.

The simplest solution would be to disallow the propagation of a WF type change to all WF instances whose execution graph was "locally" modified due to some exceptional situation. This approach could be simply handled, but it is rather unsuited for the support of long-running processes [RHD98]. Another solution would be to completely reapply all formal checks for each of these WF instances. For performance reasons, however, this approach would be disadvantageous, especially if a large number of instances have to be adapted. The challenge, therefore, is to provide efficient mechanisms, which enable the system to propagate WF type changes to a large number of WF instances, independently from whether they possess a process structure that differs from that of their original schema or not. To achieve this goal, for all WF aspects that may be subject of a change, proper conflict relations have to be defined. On top of this, it must be possible to detect potential conflicts between concurrent changes by means of simple conflict tests. For example, the WfMS must be able to efficiently check, under which conditions conflicting changes of the data flow of a WF instance may lead to inconsistencies or errors in the sequel. Having a closer look at the nature of ad-hoc changes, however, this approach is not as simple as it looks like at first glance. The reason for this is that ad-hoc changes of a WF instance graph may have a different durability. While some of them may be permanently valid until the termination of the instance, others may be only of temporary nature and must therefore be removed from the WF instance graph at the occurrence of corresponding events (e.g., when a loop back is performed).

3   Scalability Issues in Adaptive Workflow Systems

Enterprise-wide and cross-enterprise WF-based applications are characterized by a large number of users and concurrently active WF instances. As a consequence the WF servers have to cope with a very high load. Already the processing of a single WF activity may require the transfer of multiple messages between the WF server and its clients, e.g., to transmit input and output data of the activity, to update worklists, to invoke activity programs, or to exchange application data between activity programs and external

data sources. Obviously, this amount of communication may overload both, WF servers and subnets, if the number of concurrently active WF instances increases. In addition, the organizational units, which participate in a cross-organizational WF, are often connected by slow wide area networks. The load of the communication system, therefore, is extremely critical for the performance of the system. In order to keep communication local within one network segment as far as possible, in many cases, it is advantageous to dynamically migrate the control of in-progress WF instances to a new WF server in another network segment. Apart from this, due to autonomy reasons, a business company will not always tolerate that its activities are controlled by the WfMS of a foreign company. For cross-organizational workflows this means, that they cannot be completely executed by the WfMS of a single company, but may have to be controlled by distributed, (potentially) heterogeneous WfMS.

At least for some WF classes, it would be very attractive to distribute WF control onto several servers. In the WF literature, a number of distribution models have been proposed [BaDa99a, Muth98, ShKo97]. In the ADEPT project, we have also developed such a model, which is called ADEPT$_{distribution}$ [BaDa97, BaDa98]. It supports the WF designer in partitioning a WF schema and in distributing the different partitions across several WF servers. This distribution is done in a way that prevents single system components (WF servers, network segments, and gateways) from becoming overloaded at run-time. Like other distribution models, so far, ADEPT$_{distribution}$ has assumed that the structure of a WF instance graph is not changed during run-time. As shown in Section 2, this assumption does not hold for adaptive WfMS. Which additional issues arise if dynamic WF changes have to be supported in such environments? And how can we ensure that the advantages of a distributed WF control do not get lost if dynamic changes have to be supported as well? In order to be able to discuss relevant issues, first of all, we sketch the basic ideas of the ADEPT$_{distribution}$ approach in Section 3.1. Based on this model, in Section 3.2 we discuss important issues arising from the integrated support of dynamic WF changes and distributed WF control.

3.1 Distributed Workflow Control in ADEPT$_{distribution}$

At build-time, the schema of a WF is divided into several *partitions*. Each partition is assigned to a WF server, which controls the activities of this partition during run-time. For this purpose, at each server a copy of the (complete) WF schema is stored. If a WF instance reaches a transition between two partitions, its control *migrates* to the WF server of the target partition. Before this WF server may proceed with the execution of the WF instance, WF control data and WF relevant data have to be transmitted. Activities from parallel execution branches can be controlled by different WF servers in this approach, so that more than one WF server may be involved in the current execution of a particular WF instance. In order to keep communication costs low, generally, ADEPT$_{distribution}$ does not require that these WF servers synchronize with each other. An example of a WF instance graph, which is controlled by multiple WF servers, is depicted in Figure 1. In the current state, two servers – S_2 and S_3 – participate in the processing of this instance. Note that S_2 does not know the execution state of the lower branch, i.e., it does not know whether this branch is still controlled by S_1, the control has already been migrated to S_3 (as in the example shown), or it has been given back to S_1 (in order to control the partition P_4). Conversely, S_3 has no knowledge about the processing state of the partition P_2, which is controlled by S_2.

ADEPT$_{distribution}$ partitions a WF schema in a way that minimizes the communication load of the system at run-time. For this purpose the WF designer is supported by a set of algorithms, which allow him or her to
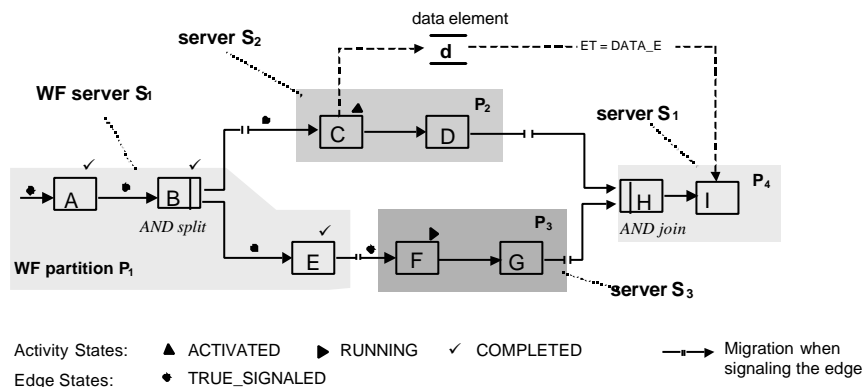

Figure 1: *Distributed execution of workflows in ADEPT*

automatically calculate optimal server assignments for the activities of a given WF schema – a partition then consists of a subgraph of which the activities are assigned to the same WF server. In doing so, it is assumed that a WF activity may be controlled by an arbitrary WF server of the WfMS, unless the WF designer explicitly excludes it from the control of this activity. To determine optimal server assignments, we use a formal cost model that allows us to evaluate the quality of a selected distribution. Among other things, this model considers costs for the transfer of parameter data, for the update of worklists, and for the migration of WF instances. In most cases, a WF activity will be controlled by a WF server, which is located nearby its potential actors. Since migrations do also generate communication costs, however, they are only used if they improve the communication behavior of the system. Details on this work and descriptions of the developed algorithms for partitioning WF schemes can be found in [BaDa98]. These algorithms do also consider so-called *variable server assignments*. Unlike static server assignments, where the WF servers that control the activities of a WF, are completely pre-defined at build-time, variable server assignments enable the WfMS to select the WF server of a particular WF activity dynamically, depending on the control data of preceding activity executions[2] (see [BaDa99b] for details). This approach contributes to improve the communication behavior of the system, especially if *dependent actor assignments*[3] are used [BaDa98, BaDa99b].

3.2   Dynamic Workflow Changes and Distributed Workflow Execution

In the ADEPT project [DaRe98] we have developed the ADEPT$_{flex}$ calculus, which provides a complete and minimal set of change operations for dynamic WF modifications in process-oriented WfMS [ReDa98, RHD98]. Most of the concepts offered by ADEPT$_{flex}$ have been prototypically implemented in the ADEPT-WfMS. ADEPT$_{flex}$ uses the same formal WF meta model as the ADEPT$_{distribution}$ approach described in the previous section. So far, ADEPT$_{distribution}$ does not consider aspects related to dynamic WF changes. Conversely, up to now ADEPT$_{flex}$ has assumed that a WF instance is controlled by one central WF server. From a logical point of view, this assumption is helpful for verifying the correctness of a change [ReDa98]. The deletion of a WF activity from a WF instance graph, for example, may lead to missing input data of subsequent activities. In order to preserve a proper data flow, these data dependent steps either have to be deleted as well (cascading deletion) or the correctness of the data flow specification has to be restored by additional adaptations (e.g., by adding so-called data provision steps to the WF instance graph) [RHD98]. If a WF is controlled by multiple servers, such a change may affect more than one server. In the WF instance graph from Figure 1, for example, the deletion of activity C (change within the partition P$_2$) would entail concomitant changes of the partition P$_4$, like the deletion of the data-dependent activity I or the addition of a preceding data provision step to I. Generally, a structural change may affect several partitions of a WF instance graph that may be controlled by different servers.

How can dynamic WF changes be supported in a distribution model like ADEPT$_{distribution}$, without loosing the advantages offered by the distributed control of workflows? First of all, it is important to minimize the synchronization effort, which becomes necessary when a dynamic change is performed. A naive solution would be to synchronize all servers that have already been involved in the processing of the WF instance or that may become active in the future. Generally, such a strict synchronization is not required and – in the case of variable server assignments (cf. Section 3.1) – it is also not possible. Instead, it is more favorable to synchronize only those WF servers, which are currently involved in the processing of the instance. If a WF instance graph does not contain AND-splits (i.e., there are no parallel execution branches), at each point in time only one WF server is responsible for the control of this instance. Consequently, no additional synchronization effort results. For parallel execution branches, however, several servers may be concurrently involved, so that a dynamic WF change may require a synchronization between them.

We will illustrate this by a simple example. Taking the change operations provided by ADEPT$_{flex}$ [ReDa98] and the WF instance graph from Figure 1, it is possible to dynamically insert a new activity X into this graph, of which the execution may not start before step F is completed and must terminate before activity D can be activated. Internally, this change is realized by the application of a well-defined set of graph transformation and graph reduction rules [ReDa98]. After its insertion, the step X constitutes a new branch of the parallel branching defined by the AND-split B and the corresponding AND-join H. The

---

[2] An example of a variable server assignment may be "*Server(A$_2$):= Domain(Actor(A$_1$))*". This means that activity A$_2$ is dynamically assigned to the server that is located in the domain of the actor who worked on activity A$_1$.

[3] With this, we mean logical assignments like "Activity A$_2$ must be executed by the same actor who has worked on the preceding activity A$_1$".
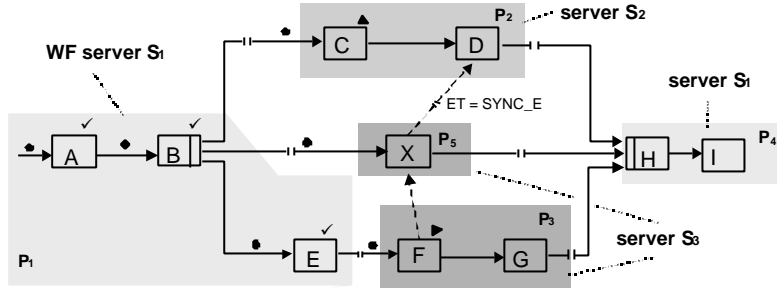
Figure 2: *The same instance graph after performing a dynamic change.*

desired execution order (X after F, X before D) is enforced by the additional insertion of the two synchro-nization edges[4] F → X and X → D (cf. Figure 2). Assume that this change is initiated by a client con-nected to the WF server S_3. Obviously, the desired modifications are only allowed, as long as D has not been started. In order to check this, first of all, the WF server S_3 must retrieve the current state of activity D from the WF server S_2 (Note that S_3 itself does not know the state of the upper branch). Conversely, the change must not only be applied to the WF instance graph stored at S_3, but it must also be considered for the copy of this graph stored at S_2. The latter becomes necessary in order to ensure that S_2 delays the execution of D until the newly inserted step X will be completed. WF servers that may become active in the future (like S_1 in our example) must not be immediately notified about the change. Instead, it is suffi-cient to transmit the corresponding information, when the control of the WF instance migrates to this server. This approach, however, causes additional communication costs for "normal" migrations, which should be kept as minimal as possible.

For distributed workflows the correct handling of dynamic WF changes is a non-trivial problem. In the following, we discuss important issues that arise in connection with the approach described above:

- How must a migration be performed, if the corresponding WF instance graph has been changed? – As already mentioned, the target server of a migration may only possess an old version of the schema of the WF instance. On the one hand, it should be avoided that a complete description of the modified WF instance graph is transferred to the target server. On the other hand, the new process structure must be made available at this server, in order to correctly proceed with the flow and to provide a proper basis for subsequent changes. The use of execution and change histories, which already exist in ADEPT$_{flex}$ [ReDa98], offers a promising approach for reducing the communication amount. In the ex-ample from Figure 2, when migrating the control from the S_2 (or S_3) to S_1 (S_1 controls the partition P_4), in addition to WF control data and WF relevant data, the corresponding entries from the change his-tory have to be transmitted as well. The WF server S_1 must then apply the modifications to its local copy of the WF instance graph, before the execution may proceed. As far as possible, the redundant transfer of information about a change should be avoided.

- Is it possible to perform a dynamic WF change without synchronizing all WF servers currently in-volved in the control of the WF instance? – Similarly to the execution of parallel branches, it is desir-able to perform dynamic changes of single branches without costly communication with other WF servers. As shown in the example, for changes that affect multiple partitions of a WF instance graph this will not always be possible. Instead, it must be ascertained for how long and in which mode the WF instance has to be locked at the respective WF servers. As far as possible, long-term locks should be avoided, so that the WF execution is not blocked unnecessarily long. There are numerous examples for dynamic WF changes, for which such a strict synchronization does not become necessary. Taking the WF instance graph from Figure 1, for example, the WF server S_2 might insert a new activity be-tween C and D or delete the activity C (incl. the deletion of the outgoing data edge connected to d and the deletion of the data-dependent activity I) without synchronizing this change with S_3. This is possi-ble, since S_2 does not require information about the state of the lower branch in order to apply the change. Conversely, S_3 must not be informed about the modification of the upper branch in order to proceed with the control. Such local modifications do occur often in practice. Therefore it does make

---

[4] Synchronization edges are a special edge type of our graph-based WF meta model ADEPT$_{base}$ [ReDa98]. They can be used for modeling different kinds of "wait-for" situations between activities from parallel execution branches.

sense to differentiate between different classes of changes with respect to a WF instance graph (e.g., local change of a partition, changes of several partitions controlled by the same WF server, changes of partitions controlled by different WF servers, etc.) and to offer optimized procedures for their application. Generally, if a WF server wants to perform a dynamic change, it must determine which information have to be retrieved from which other servers to perform the desired modifications and which servers must be informed about the change afterwards.

- Which adaptations will become necessary regarding server assignments, if a WF instance graph is structurally changed? – For newly inserted activities, for example, appropriate WF servers must also be assigned to. On the one hand, it seems to be attractive to use the same distribution algorithms that are applied at build-time [BaDa98] (Note that this may lead to changed server assignments of other WF activities as well). On the other hand, if these run-time calculations are too costly, simpler approaches for the (dynamic) assignment of servers may be favorable. In the example from Figure 2, the newly inserted activity X has been assigned to the WF server S_3, which initiated the change.

- Assume that a WF server S, which wants to apply a dynamic change to a WF instance, controls a particular partition from a parallel branch of the WF instance graph. How can this WF server find out, which other WF servers are currently involved in the processing of this WF instance (see above)? – The main problem in this context arises from the fact that, generally, S has no information about the state of activities from parallel branches (cf. Section 3.1), unless S controls these activities itself or has obtained additional information when an incoming synchronization edge was signaled [ReDa98]. The server S does also not know, whether the processing of a parallel branch is in a state before or after a migration. Using the information locally available, S does not know, which other WF servers are currently active. With respect to the WF instance graph from Figure 1, this applies to the WF server S_2, for example. This server does not know, whether the lower branch is currently controlled by S_1 or S_3. This problem even gets worse if variable server assignments (cf. Section 3.1) are used.

- Which additional problems arise in connection with variable server assignments? – If the server selection of an activity A_2 depends on the execution of a preceding activity A_1 (cf. Section 3.1), it must be ensured that a WF server can be assigned to A_2, even if A_1 will be dynamically removed from the WF instance graph. In this context, it seems to be attractive to apply similar exception handling mechanisms, as they have been used in ADEPT$_{flex}$ to guarantee the provision of activity input parameters in the case of structural WF changes [ReDa98].

Although we have used the ADEPT workflow model for illustration purposes, most of the issues discussed, do also apply to other flexible WF models.

4   Further Issues

We shortly discuss two other basic requirements, which are essential for the flexible support of large-scale WF-based application systems.

4.1   Semantic Rollback of Workflows in Adaptive WfMS

An adaptive WfMS must offer adequate concepts to cope with semantic failures of single WF activities at run-time. During the last years, a number of advanced transaction concepts have been developed [Alon96, JaKe97, Leym95]. With few exceptions [LiPu98, MüRa99], however, most of them are based on the assumption that the flow structure of the transaction or the workflow, respectively, is completely known at build-time. As shown in Section 2, this assumption is only valid for completely static workflows and does not apply to adaptive WfMS. The proposed concepts are therefore not suited for the flexible support of a broader spectrum of workflows.

Normally, for the transactional support of long-running processes, we cannot strictly enforce the atomicity of the whole composite transaction (workflow). Instead, intermediate results may become visible and may be modified by other transactions. As a consequence ordinary transaction rollback is no longer applicable in this context. An extended transaction mechanism must allow the WF designer to define compensation steps for WF activities in order to enable the WfMS to support some kind of "logical rollback" for already completed and committed activities, if semantic failures occur at run-time [JaKe97]. One important problem in this context, which has not been satisfactorily solved in the literature so far, arises from the fact that the kind of compensation of a WF step may depend on the current state of the WF instance, on the point in time a failure occurs, or on other factors. Obviously, these dependencies compli-

cate the definition of appropriate "recovery spheres" [Leym95] and they also complicate application development. It will be one of the key factors for the success of adaptive WF technology, whether it will be possible to develop sophisticated concepts for describing compensation spheres, for implementing WF activities and their (perhaps different) compensation steps, and for dealing with dynamic cases and dynamic WF changes (see above). In connection with dynamic changes, among other things, it must be considered that the definition of compensation spheres may have to be adapted when the structure of the WF instance graph is dynamically modified.

In addition to these requirements, for enterprise-wide and cross-enterprise workflows numerous other problems have to be solved. It has to be specified, for example, under which conditions an actor is authorized to reset WF activities that are controlled by the WfMS of a foreign company. In this context, we must not only deal with technical issues (e.g., support of different commit protocols), but we must also consider numerous other aspects (privacy, documentation, differences in law for companies from different countries, etc.). All these issues must be carefully analyzed and understood, taking into consideration all the non-trivial interdependencies with other features of the system, like e.g., the support of different kinds of changes (cf. Section 2) or the distributed execution of workflows (cf. Section 3).

4.2   Security Issues in Adaptive WfMS

Very often, a WfMS processes data for which high standards must be set with respect to privacy and data security. Generally, in a WfMS the access rights of a person are determined by the roles or functions he or she may take. Already for the static ("unflexible") case, however, the definition of appropriate roles and organizational rules for substitutions may become very complex and poses high requirements for the WfMS, especially when looking at the maintenance of organizational models. For the "flexible" case, in addition, we must ensure that dynamic changes of a WF instance do not lead to "security gaps". For cross-organizational workflows we have to deal with the difficulty that WF changes, which are reasonable from the point of view of a single organizational unit, may be in conflict with superior process goals (e.g., temporal constraints, quality requirements, etc.). For these reasons, it must be possible to control, at a very fine level of granularity, which persons or roles may perform which changes with respect to a particular WF instance or with respect to instances of a particular WF type.

Most of the challenges discussed in the previous sections apply to "normal" WF applications as well. Nevertheless, in the context of enterprise-wide and cross-enterprise workflows they have a special relevance and an extreme importance.

5   Summary and Outlook

To be applicable to enterprise-wide and cross-enterprise workflows, a process-oriented WfMS must cover a broad spectrum of processes and it must support different kinds of dynamic changes at run-time. These changes must be accomplished in an efficient and secure manner and without affecting the robustness of the system. This must also apply if the number of users and concurrently active WF instances is high. In this context, it is important that the "normal" WF execution is not affected too much by the additional features of an adaptive WfMS. In the ADEPT project the described challenges have been exhaustively investigated. Formal issues and the basic principles of dynamic WF changes are well understood in the meantime. Furthermore they have been prototypically implemented in the ADEPT-WfMS [ReDa98, RHD98]. We also work on issues related to WF schema evolution and their interrelation with ad-hoc deviations. The same applies to scalability issues [BaDa97, BaDa98, BaDa99a] as well as to some other aspects (for an overview see [DaRe98]). An important next step will be, to interweave the different concepts and to realize common implementations.

References

[Alon96]   Alonso, G.; Agrawal, D.; El Abbadi, A.; Kamath, M.; Günthör, R.; Mohan, C.: *Advanced Transaction Models in Workflow Contexts*. Proc. 12th Int'l Conf. on Data Engineering. New Orleans, February 1996

[BaDa97]   Bauer, T.; Dadam, P.: *A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration*. Proc. CoopIS '97, Kiawah Island, June 1997, pp. 99-108

[BaDa98]   Bauer, T.; Dadam, P.: *Variable Migration of Workflows in ADEPT* (in German). Ulmer Informatik-Berichte, Nr. 98-09, September 1998

[BaDa99a]   Bauer, T.; Dadam, P.: *Distribution Models for Workflow Management Systems – Classification and Simulation* (in German). Ulmer Informatik-Berichte, Nr. 99-02, University of Ulm, March 1999

[BaDa99b]   Bauer, T.; Dadam, P.: *Efficient Distributed Control of Enterprise-Wide and Cross-Enterprise Workflows.* Proc. Workshop Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI (Informatik'99). Paderborn, October 1999

[CCPP98]   Casati, F.; Ceri, S.; Pernici, B.; Pozzi, G.: *Workflow Evolution*. Data & Knowledge Engineering, 24(3), Jan. 1998, pp. 211-238

[DaRe98]   Dadam, P.; Reichert, M.: *The ADEPT WfMS Project at the University of Ulm.* Presented at the 1st European Workshop on Workflow and Process Management (WPM'98) – "Workflow Management Research Projects", Zürich, October 1998

[JaKe97]   Jajodia, S.; Kerschberg, L. (Eds.): *Advanced Transaction Models and Architectures.* Kluwer Academic Publishers, 1997

[JoHe98]   Joeris, G.; Herzog, O.: *Managing Evolving Workflow Specifications.* Proc. CoopIS'98, New York, August 1998.

[Leym95]   Leymann, F.: *Supporting Business Transactions via Partial Recovery in Workflow Management Systems*, Proc. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW '95), Dresden, March 1995, pp. 51-70

[LiPu98]   Liu, L.; Pu, C.: *Methodical Restructuring of Complex Workflow Activities.* Proc. 14th Int'l Conf. On Data Engineering (ICDE'98), Orlando, Florida, February 1998, pp. 342-350.

[MüRa99]   Müller, R.; Rahm, E.: *Rule-Based Dynamic Modification of Workflows in a Medical Domain.* Proc. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW '99), Freiburg, March 1999, pp. 429-448.

[Muth98]   Muth, P.; Wodtke, D.; Weißenfels, J.; Kotz-Dittrich, A.; Weikum, G.: *From Centralized Workflow Specification to Distributed Workflow Execution.* Journal of Intelligent Information Systens, Special Issue on Workflow Management Systems, Vol. 10, March/April 1998, pp. 159-184

[ReDa98]   Reichert, M.; Dadam, P.: *ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control.* Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, Vol. 10, March/April 1998, pp. 93-129

[RHD98]   Reichert, M.; Hensinger, C.; Dadam, P.: *Supporting Adaptive Workflows in Advanced Application Environments.* Proc. EDBT-Workshop on Workflow Management Systems, Valencia, March 1998, pp. 100-109

[ShKo97]   Sheth, A.; Kochut, K.: *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems.* Proc. NATO Advanced Study Institute on Workflow Man Sys and Interop. Istanbul, August 1997, pp. 12-21

[Sieb98]   Siebert, R.: *An Open Architecture for Adaptive Workflow Management Systems.* Proc. 3rd Biennial World Conf on Integrated Design and Process Techn, Vol 2, Berlin, July 1998, pp. 79-85.

[Wesk98]   Weske, M.: *Flexible Modeling and Execution of Workflow Activities.* Proc. 31st Hawaii Int'l Conf. on System Sciences (HICSS–31), Software Technology Track (Vol VII), 1998, pp. 713–722

Workflows over Workflows:
Practical Experiences with the Integration of SAP R/3 Business Workflows in WISE[*]

Christoph Schuler   Heiko Schuldt   Gustavo Alonso   Hans-Jörg Schek

Institute of Information Systems
Swiss Federal Institute of Technology (ETH)
ETH Zentrum
8092 Zürich, Switzerland
Email: {schuler,schuldt,alonso,schek}@inf.ethz.ch

Abstract

Business processes within companies are in general well established and supported by commercially available workflow management systems. However, when processes span multiple companies (such as, for instance, in the case of virtual enterprises or in business to business electronic commerce), software support quite is limited. In such cases, processes need to be implemented on top of already existing systems. The WISE project addresses this problem and provides adequate infrastructure to support the whole life-cycle of virtual enterprise processes, including the integration of legacy systems, especially of workflow management systems. In this paper, we describe our experience with the integration of SAP R/3 Business Workflows into WISE. In addition, and since an important characteristics of the WISE system is the enactment of processes with execution guarantees, we also discuss the provision of execution guarantees when external systems like SAP R/3 Business Workflows are involved.

1 Introduction

In a *virtual enterprise*, different companies temporarily work together in common projects to achieve common goals. In order to support this collaboration, the necessary infrastructure has to be available. In general, business processes within each company can be supported by workflow management systems. In virtual enterprise environments, however, processes go beyond corporation boundaries and encompass services provided by the different participants. A workflow management system supporting virtual business processes in a virtual enterprise must now face the challenge of incorporating existing processes running with a variety of workflow management systems.

The WISE project [AFH+99] addresses this problem and provides an infrastructure to support the execution of virtual business process in virtual enterprises. As part of the WISE project, we have been working on the integration of workflow processes running in different workflow management systems such as, for instance, SAP R/3 Business Workflow and IBM FlowMark, into the WISE system. This

---

extended abstract focuses on the integration of SAP R/3 Business Workflows, discusses the approach taken, and presents practical experiences made during this endeavor.

The paper is structured as follows: In Section 2, we present the WISE system, a workflow management system developed to support viable business to business Electronic Commerce. Then, in Section 3, we provide a brief overview of the interoperability problems that have to be addressed when embedding legacy applications in workflow processes while at the same time providing execution guarantees for these processes. We then focus in Section 4 on the practical experiences gained by integrating SAP R/3 Business Workflow processes into WISE. Section 5 finally concludes the paper.

2 WISE: **Workflow based Internet SErvices**

The WISE project [AFH$^+$99] aims to provide a viable infrastructure for business to business electronic commerce in virtual enterprises. To this end, existing services of companies temporarily participating in a virtual enterprise are linked together and embedded within a virtual business process. To do this, WISE implements four different components: *definition*, *enactment*, *monitoring*, and *ad-hoc coordination*.

The *definition* component provides the possibility to create virtual business processes supported by an appropriate graphical interface (IvyFrame, a commercial business modeling tool by IvyTeam [Ivy]). The *enactment* component compiles these process models in a format suitable for execution and controls the execution of virtual business processes. In order to keep track of the state of virtual business processes, the *monitoring* component extracts and visualizes the necessary information. The *coordination* component finally offers the participants of a virtual business process the possibility to establish multimedia conferences based on the information produced during execution.

3 **Integration of Applications into** WISE

The typical processes implemented in WISE encompass activities which are invocations of different subsystems (e.g. processes running in different workflow management systems). One key feature of the WISE engine is to provide execution guarantees for these processes even in case of failures and of concurrent access to shared resources. To this end, the ideas of transactional process management [SAS99] are applied. Among others, these execution guarantees include guaranteed termination, a more general notion of atomicity than the standard all or nothing semantics. Guaranteed termination is realized by partial compensation and alternative executions. In addition, WISE also controls the parallelization of concurrent processes to guarantee correct interleavings.

3.1 **Transactional Coordination Agents (TCAs)**

In order to enforce such execution guarantees by the WISE engine, each participating subsystem must meet a series of requirements. These requirements include the following database functionality for single activities: All activities have to be atomic to avoid inconsistencies due to the undefined outcome of non-atomic activities within a subsystem. In case that activities can be semantically compensated once they have been executed correctly, the availability of this compensation has to be guaranteed. When activities can not be compensated, their commit may have to be deferred in certain cases. Finally, for some activities a repeated invocation with "exactly-once-semantics" is necessary in order to guarantee their successful execution. Furthermore, when parallel access to shared resources takes place, orders established between activities by the WISE engine have to be respected in the underlying subsystems.

Since the goal is to integrate arbitrary applications into WISE (and especially arbitrary workflow management systems), these requirements are in general not met. Therefore, each participating subsystem is wrapped by a transactional coordination agent (TCA) [NSSW94, SST98]. The WISE system then acts as a process scheduler on top of several TCAs serving as lower level schedulers. The task of the TCA that has to be provided for each application is therefore not only to exploit the subsystem specific interfaces to allow local services to be invoked from WISE but also to provide the required database functionality on top of the application (a detailed discussion can be found in [SSA99]). Thus, TCAs extend the idea of the application agents defined within the workflow reference model [Hol93] of the Workflow Management Coalition (WfMC) by providing additional database functionality on top of application systems.

3.2 Structure of Generic TCA

With respect to the functionality to be provided by each TCA, there are four different modules to consider [Wun96]: communication, scheduling, monitoring and execution (figure 1).

To support *execution*, activities must be mapped to local operations. For this purpose, subsystem-specific interfaces have to be exploited and thus, the execution module has to be tailored to the application to be integrated.

Furthermore, scheduling of local operations with respect to the activities specified by the WISE system has to be performed by the *scheduling* module. This includes the preservation of the given orders as well as the provision of local atomicity, the guaranteed availability of compensating activities, and eventually the deferment of local commits. For these purposes, the TCA has to persistently log the activities executed within the subsystem and eventually also the activities needed for compensation purposes.

In order to support the interaction between the WISE system and TCAs, a common communication protocol has to be provided by the *communication* module.

Figure 1: Structure of a generic TCA

Finally, the *monitoring* module covers the extraction of local state information from the underlying application. For this purpose, again existing interfaces have to be exploited such that the monitoring module also has to be tailored to the subsystem.
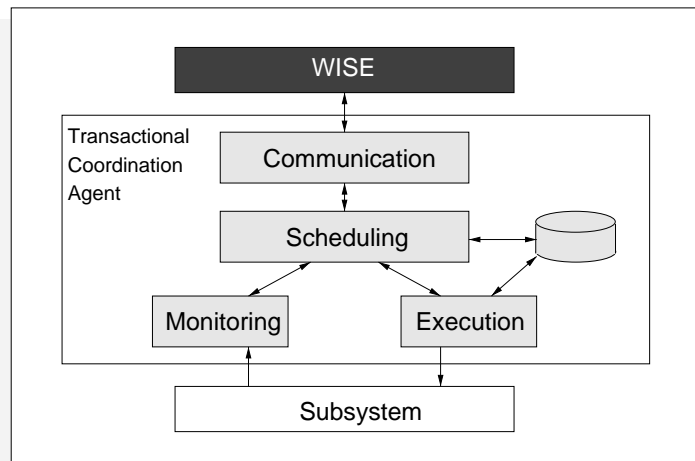
4 Integration of SAP Business Workflows

SAP R/3 [SAP] is one of the most commonly used application systems for business management purposes. It consists of specialized modules for several application areas (e.g., production planning, logistics, human resource management, or controlling). The system is built in a client/server architecture and is based on a relational DBMS [BEG96].

In order to integrate arbitrary SAP R/3 Business Workflows [WFB+95, SAP96] as part of WISE processes, a transactional coordination agent for SAP R/3 has been implemented [Sch98]. The architecture of this TCA is depicted in figure 2. The agent specific parts are colored in light gray whereas the standard SAP R/3 system is depicted in white color. The agent is tightly integrated into the core system as, for instance, R/3's underlying database is used for the management of the TCAs' metadata and is accessed through the standard DBMS interface of SAP R/3. The kernel of the TCA is itself a SAP workflow process (meta process) in which the function or the workflow to be executed is embedded in a generic way by simply passing its name as parameter to the meta process. In what follows, we describe in detail the architecture of this SAP R/3 agent.
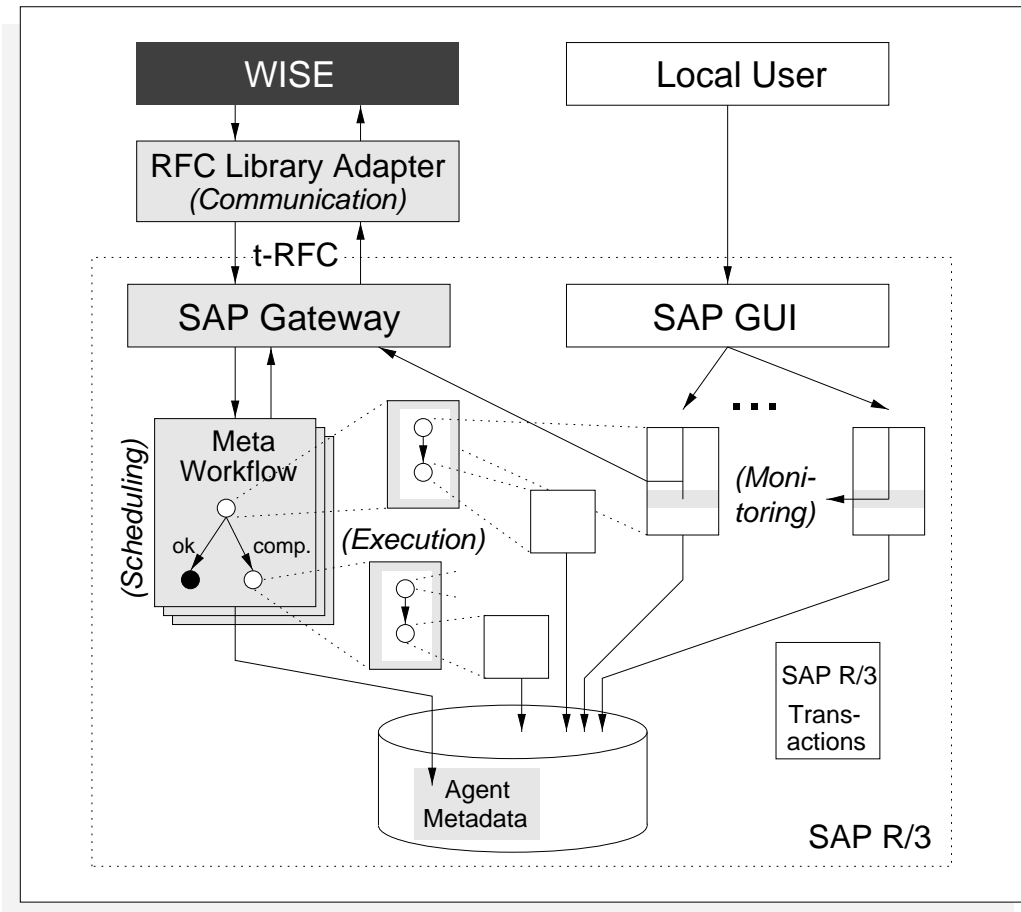

Figure 2: Architecture of the SAP R/3 agent

For execution purposes, the meta workflow of the TCA is called with a specification of the name of the function or process to be executed.

Communication with the WISE system takes place by exploiting the transactional remote function call (t-RFC) of SAP R/3 (t-RFC is the transactional variant of SAP's implementation of the remote procedure call, RPC, which is available via a set of C library functions). It supports both multiple parallel calls of R/3 functions from external applications as well as call-backs of external applications from within the R/3 system via the SAP gateway. The communication module (t-RFC library adapter) is thus a thin software layer transforming requests from WISE into t-RFC calls and vice versa.

The monitoring task is implemented by customization of the necessary R/3 transactions and processes. This is possible since certain hooks (customer exits) are foreseen in R/3 to add user-specific code and since the ABAP/4 [KW97] sources of all R/3 transactions are available (ABAP/4, Advanced Business Application Programming Language, is the 4th generation programming language in which the greatest part of the R/3 system —aside of a small C kernel— is implemented in).

SAP R/3 supports the notion of transactions and guarantees full ACID properties for them. When a single function has to be executed as an activity in WISE, atomicity is provided by the R/3 system. When a complete workflow process has to be executed, however, appropriate failure handling mechanisms need to be implemented within this process to do a rollback in the case of failures in order to guarantee the required all-or-nothing semantics of activities.

In general, the meta workflow encompasses only two activities: the function or process to be executed and its associated compensation. In this, we follow the idea of an explicit registration of the compensation which is advantageous in that all instances of SAP R/3 workflow processes together with the associated parameters are stored persistently in the underlying DBMS. Thus, no additional effort has to be taken to log the parameters associated with an activity executed by WISE. After an activity of WISE (which corresponds to a workflow process of SAP) within the meta workflow has been executed successfully, the meta process performs an idle wait (which does not consume any resources). If the nested SAP process needs to be compensated, an internal event in SAP R/3 is generated via a t-RFC library call and the next activity of the meta workflow, the compensation, can be executed without explicitly specifying the required parameters. Otherwise, when compensation no longer has to be considered, the meta process is terminated. The structure of the meta workflow, which forms the body of the TCA, is depicted in figure 3 using the event-controlled process chain notation of SAP. Although this process conceptually encompasses only two activities, a couple of internal checks and events are relevant. First, the meta workflow writes a log entry to prevent multiple starts of the same task. After this, the desired workflow is started while in the meantime, the meta workflow waits in parallel on its termination signal (a). If an error occurred and the workflow is marked to be repeatable, the flow goes back to start the workflow again (b). After successful termination or abort of the workflow, the result is called back to the WISE system (c). The parallel section (d) communicates with the external WISE system. At this point the meta workflow waits in an idle wait until a decision is sent from the WISE system in order to compensate (e) or not.

The retriability of SAP processes is achieved by exploiting SAP R/3's transactional remote function call mechanisms. After the t-RFC
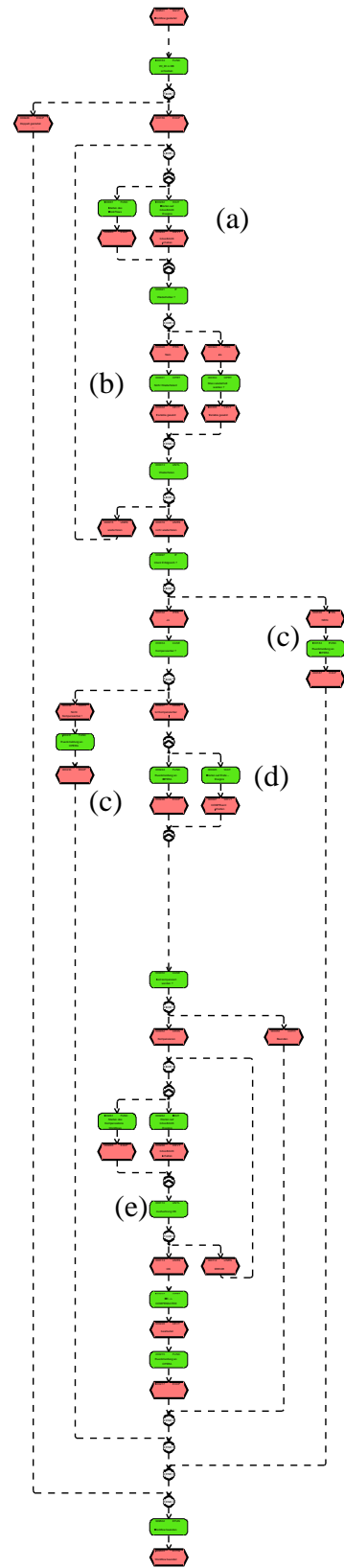
Figure 3: Structure of the agent's meta workflow in EPC notation

library adapter of the TCA has requested a t-RFC identifier from the R/3 system, all invocations of a retriable activity are performed with this ID and the R/3 system guarantees that is is executed only once even when multiple invocations of this activity occur.

SAP R/3's t-RFC was originally designed to start a function on another R/3 system. In order to invoke a workflow, a thin layer of SAP-R/3 functions is needed to propagate requests from outside to the SAP workflow management system. For this purpose, we use ABAP/4 functions in order to propagate calls from the outside world to the workflow component of SAP R/3.

This function layer also implements the mapping from the external ID used in WISE for each activity to SAP's internal workflow identification number. Once a meta workflow has been started it handles all the messages from the WISE system. For every type of command, a function is provided to send this command to the meta workflow by an internal event.

The start command invokes an instance of this meta workflow, passing the name and corresponding parameters to the workflow process to be invoked. The meta workflow marshalls the parameters and starts the desired R/3 workflow as if it was started directly from within the system. All parameters of a SAP workflow process are stored in a so-called container, which is an array of parameter name and value pairs. Since the meta workflow has to store the container of the workflow, this container has to be packed into the container of the meta workflow. That way every built-in SAP workflow can be invoked without changes on its definition. Furthermore, since only standard components of SAP R/3 are exploited, the TCA implementation will not be affected by changes introduced by new releases (it is, for instance, guaranteed that all customer exits of R/3 are also present in future versions of the system).

After the workflow terminates, the meta workflow sends a notification back to the WISE system and stays in idle wait until it can terminate itself or until it has to start the compensation workflow in order to undo the first workflow. The whole container of the workflow in the end state of the workflow is passed through meta workflows container in order to preserve all informations to undo the changes. This way we can store all this information persistently in the meta workflow's container.

The workflow model of SAP R/3 is designed to reach a consistent state after the termination of a process. As in our case we have have an additional workflow manager above SAP, we need a mechanism to propagate some result state back. To provide this feature, we introduced a special variable WF_RESULT which represents the result of the workflow execution.

5 Conclusion

In this short paper, we have discussed the practical experiences gained from the integration of SAP R/3 Business Workflows into the WISE system. This integration not only involves the invocation of SAP processes from WISE but also the provision of execution guarantees for SAP processes which can be exploited by WISE. To this end, a transactional coordination agent (TCA) is required, which is in the case of the SAP R/3 implementation tightly integrated into the system. Together with a TCA that has been implemented for IBM's FlowMark [IBM94], the SAP R/3 TCA complements the effort to allow the WISE system to run processes on top of processes.

Based on the TCAs already implemented, our future work aims in providing further support with respect to the monitoring of nested processes that are, in general, not known to WISE. To this end, an abstract description of nested processes will be made available. The TCA then publishes internal states of nested processes with respect to this abstract description such that it can be displayed by the monitoring facilities of WISE.

References

[AFH⁺99] G. Alonso, U. Fiedler, C. Hagen, A. Lazcano, H. Schuldt, and N. Weiler. WISE: Business to Business E-Commerce. In *Proceedings of the 9th International Workshop on Research Issues in Data Engineering. Information Technology for Virtual Enterprises (RIDE-VE'99)*, pages 132–139, Sydney, Australia, March 1999.

[BEG96] R. Buck-Emden and J. Galimow. *SAP R/3 System: A Client/Server Technology*. Addison-Wesley, 1996.

[Hol93] D. Hollingsworth. *Workflow Management Coalition: The Workflow Reference Model*. Workflow Management Coalition, December 1993. Document TC00-1003.

[IBM94] IBM. *FlowMark for OS/2: Managing Your Workflow*, first edition, May 1994.

[Ivy] IvyTeam, Zug, Switzerland. `http://www.ivyteam.com`.

[KW97] R. Kretschmer and W. Weiss. *Developing SAP's R/3 Applications with ABAP/4*. Sybex, 1997.

[NSSW94] M. Norrie, W. Schaad, H.-J. Schek, and M. Wunderli. CIM Through Database Coordination. In *Proceedings of the International Conference on Data and Knowledge Systems*, May 1994.

[SAP] SAP AG, Walldorf, Germany. `http://www.sap.com`.

[SAP96] SAP AG, D-69185 Walldorf. *System R/3: SAP Business Workflow*, 1996. White Paper.

[SAS99] H. Schuldt, G. Alonso, and H.-J. Schek. Concurrency Control and Recovery in Transactional Process Management. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'99)*, pages 316–326, Philadelphia, Pennsylvania, USA, May 31-June 2 1999.

[Sch98] C. Schuler. Design and Development of a Coordination Agent for the Integration of SAP R/3 in Workflow Processes. Diploma thesis, Database Research Group, Institute of Information Systems, ETH Zürich, July 1998. In German.

[SSA99] H. Schuldt, H.-J. Schek, and G. Alonso. Transactional Coordination Agents for Composite Systems. In *Proceedings of the 3rd International Database Engineering and Applications Symposium (IDEAS'99)*, pages 321–331, Montréal, Canada, August 1999.

[SST98] H. Schuldt, H.-J. Schek, and M. Tresch. Coordination in CIM: Bringing Database Functionality to Application Systems. In *Proceedings of the 5th European Concurrent Engineering Conference (ECEC'98)*, pages 223–230, Erlangen, Germany, April 1998.

[WFB⁺95] H. Wächter, F. Fritz, A. Berthold, B. Drittler, H. Eckert, R. Gerstner, R. Götzinger, R. Krane, A. Schaeff, C. Schlögel, and R. Weber. Modellierung und Ausführung flexibler Geschäftsprozesse mit SAP Business Workflow 3.0. In F. Huber-Wäschle, H. Schauer, and P. Widmeyer, editors, *GISI 95 – Herausforderungen eines globalen Informationsverbundes für die Informatik*, Informatik Aktuell, pages 197–204. Gesellschaft für Informatik (GI) und Schweizer Informatiker Gesellschaft (SI), Springer-Verlag, 1995. In German.

[Wun96] M. Wunderli. *Database Technology for the Coordination of CIM Subsystems*. PhD thesis, Swiss Federal Institute of Technology Zürich, 1996.