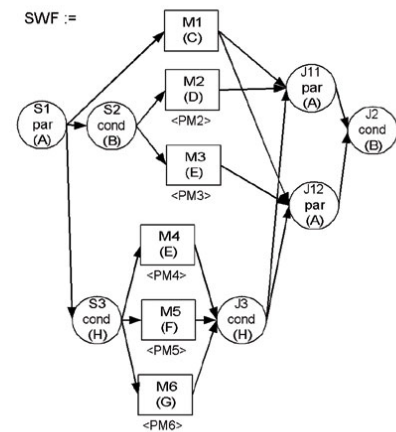


Modeling and Transformation of Workflows with Temporal Constraints



*Dissertation von
Dipl.-Ing. Wolfgang L. Gruber
Universität Klagenfurt
Mai 2003*

Dipl.-Ing. Wolfgang L. Gruber

Modeling and Transformation of Workflows with Temporal Constraints

DISSERTATION

zur Erlangung des akademischen Grades
Doktor der Technischen Wissenschaften

Universität Klagenfurt
Fakultät für Wirtschaftswissenschaften und Informatik

- | | |
|-------------------|---|
| 1. Begutachter: | O. Univ.-Prof. Dipl.-Ing. Dr. Johann Eder |
| Institut: | <i>Universität Klagenfurt</i> |
| 2. Begutachterin: | O. Univ.-Prof. Dipl.-Ing. Mag. Dr. Gerti Kappel |
| Institut: | <i>Technische Universität Wien</i> |

Mai 2003

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Schrift verfasst und die mit ihr unmittelbar verbundenen Arbeiten selbst durchgeführt habe. Die in der Schrift verwendete Literatur sowie das Ausmaß der mir im gesamten Arbeitsvorgang gewährten Unterstützung sind ausnahmslos angegeben. Die Schrift ist noch keiner anderen Prüfungsbehörde vorgelegt worden.

Klagenfurt, am 19. Mai 2003

(Dipl.-Ing. Wolfgang L. Gruber)

Acknowledgements

This thesis would not have been possible without the help and support of many people.

Therefore, I wish to thank my supervisors, Prof. Johann Eder and Prof. Gerti Kappel. Both devoted a lot of their time to discuss my work and to provide their invaluable advice.

I am deeply grateful to my director of thesis Prof. Johann Eder for his precious instruction. His dynamic thoughts, his broad and profound knowledge and his patient instruction were of great help to me. He spent much time to check and improve my thesis.

Thanks are also due to my colleagues who discussed this work on several occasions and who helped me with technical issues. Heinz Frank always had an answer to my LaTeX questions and kept asking me to finish this thesis.

Finally, I would like to thank Mag. art. Désirée Doujak-Gruber for the graphic design of the book cover, which is the first thing a potential reader sees and which leaves an indelible impression. I also wish to thank Mag. Simone Pansi for proof-reading this thesis and helping me to improve my English writing.

Abstract

Companies and organizations make a great effort to provide better products and better services at a lower cost, of reducing the time to market, of improving and customizing their relationships with customers and, ultimately, of increasing customer satisfaction and the company's profits. These objectives push companies and organizations into continuously improving the business processes that are performed in order to provide services or to produce goods.

Workflow technology has emerged as one of the latest technologies designed to accomplish above demands by modeling, redesign and execution of business processes. Especially in this context, workflow time management is important to timely schedule workflow process execution, to avoid deadline violations, and to improve the workflow turn-around times.

In this thesis, we proposed modeling primitives for expressing time constraints between activities and binding activity executions to certain fixed dates (e.g. first day of the month). Time constraints between activities include lower bound and upper bound constraints. In addition, we present techniques for checking satisfiability of time constraints at process build time. These techniques compute internal activity deadlines in a way that externally assigned deadlines are met and all time constraints are satisfied. Thus the risk of missing an external deadline is recognized early and steps to avoid a time failure can be taken.

Our actual work focuses on: (1) providing an advanced workflow metamodel that supports hierarchical composition of complex activities and reuse of activities in several workflow definitions; (2) modeling of time and time constraints to capture the available time information; (3) developing PERT-net based pro-active time calculations for computing internal deadlines to capture time constraint violations and raise alerts in case of potential future time violations; (4) accomplish workflow transformations in order to tackle the problem of unnecessary rejections induced by superfluous time constraint violations; and (5) describing our graphical WF-Designer prototype.

Copyright © 2003 by Wolfgang L. Gruber



Contents

Acknowledgement	v
Abstract	vii
Content	xi
1 Introduction	1
1.1 Motivation for Research on Time Management	2
1.2 Objectives and Contributions of this Thesis	3
1.3 Structure of the Thesis	4
2 Workflow Systems Overview	7
2.1 Introduction	7
2.1.1 Workflow Management System	8
2.1.2 Workflow Reference Model	9
2.1.3 Workflow Classification	10
2.2 Summary	12
3 Workflow Modeling	13
3.1 Introduction	14
3.2 Related Work	14
3.3 Workflow Definition	15
3.3.1 Conformance Classes	15
3.3.2 Structured Workflow Definition	16
3.3.3 Occurrence(s) of Activities	16
3.4 Workflow Models	16
3.5 Workflow Representation	20
3.5.1 Programming Language Style Text Based Representation . . .	20
3.5.2 Workflow Graphs	21
3.5.2.1 Strictly Structured Workflow Graph	22
3.5.2.2 Less Strictly Structured Workflow Graph	22

3.6	Workflow Control Structures	23
3.6.1	Sequence	23
3.6.2	Parallelism	23
3.6.3	Conditional	23
3.6.4	Alternative	24
3.6.5	Iteration	25
3.7	Composition Structure of the Workflow Model	25
3.8	Summary	26
4	Workflow Metamodel	29
4.1	Introduction	29
4.2	Metamodel Requirements	30
4.3	Related Work	31
4.4	Workflow Metamodel Description	32
4.4.1	Metamodel Stratifications	32
4.4.2	Workflows and Activities	32
4.4.2.1	Activity	32
4.4.2.2	Complex Activity	34
4.4.3	Occurrences	34
4.4.3.1	Activity Occurrence	35
4.4.3.2	Control Occurrence	35
4.4.4	Workflow Model	35
4.4.4.1	Model Element	35
4.4.4.2	Model Activity Occurrence	36
4.4.4.3	Model Control Occurrence	36
4.4.5	Metamodel Integrity Constraints	36
4.4.5.1	Integrity Constraints Belonging the Specification Level	38
4.4.5.2	Integrity Constraints Belonging to the Model Level	41
4.5	Workflow Example	43
4.6	Summary	44
5	Workflow Transformations	45
5.1	Introduction	45
5.1.1	Types of Workflow Changes	46
5.1.2	Process Perspective Changes	46
5.1.3	Classes of Transformation	47
5.2	Related Work	48
5.3	Workflow Instance Type	48
5.4	Equivalence of Workflows	49
5.4.1	Equivalence Definitions	51
5.4.1.1	Equivalence Relation	51

5.4.1.2	Equivalence Example	52
5.4.1.3	Equivalence Transformation Relation	53
5.5	Basic Transformations	54
5.5.1	Hierarchy Manipulation	54
5.5.1.1	Flatten/Unflatten (WFT-H1)	55
5.5.1.2	Encapsulation in a Sequence (WFT-H2)	55
5.5.2	Moving Joins	57
5.5.2.1	Join Moving Over Activity Occurrence (WFT-J1)	57
5.5.2.2	Join Moving Over Seq-Join (WFT-J2)	58
5.5.2.3	Moving Or-Join Over Or-Join (WFT-J3)	59
5.5.2.4	Moving Alt-Join Over Alt-Join (WFT-J4)	61
5.5.2.5	Moving Or-Join Over Alt-Join (WFT-J5)	62
5.5.2.6	Moving Alt-Join Over Or-Join (WFT-J6)	64
5.5.2.7	Join Coalescing (WFT-J7)	65
5.5.2.8	Separating a Conditional/Alternative Path (WFT-PS)	67
5.5.2.9	Moving Join Over And-Join - Unfold (WFT-J8)	68
5.5.2.10	And-Join Moving Over Or-Join (WFT-J9)	71
5.5.3	Split Moving	73
5.5.3.1	Moving Split Before Activity Occurrence (WFT-S1)	73
5.5.3.2	Split Moving Over Seq-Join (WFT-S2)	74
5.6	Complex Transformations	75
5.6.1	Backward Unfolding Procedure	75
5.6.2	Partial Backward Unfolding Procedure	79
5.6.3	Regarding Transformation Sequence When Unfolding	80
5.6.4	Forward Unfold Procedure	82
5.7	Application	83
5.8	Summary	83
6	Time Management in Workflows	85
6.1	Introduction	85
6.2	Related Work	86
6.3	Time Modeling in Workflows	90
6.3.1	Build-Time, Run-Time and Post-Run-Time	90
6.3.2	Time Aspects in Workflow Management Systems	92
6.3.2.1	Ontology of Time	92
6.3.2.2	Execution Durations and Deadlines	92
6.4	Time Constraints	93
6.4.1	Implicit Time Constraints	93
6.4.2	Explicit Time Constraints	94
6.4.2.1	Temporal Relationship Constraints	94
6.4.2.2	Checking Constraint Correctness	95

6.4.2.3	Fixed-Date Constraints	96
6.5	Timed Workflow Model	98
6.5.1	Timed Workflow Metamodel	99
6.6	Representations of Timed Workflow Models	101
6.6.1	PERT Chart	101
6.6.2	CPM Chart	102
6.6.3	Gantt Chart	102
6.6.4	ePERT Chart	103
6.7	Timed Workflow Graph	104
6.7.1	Constraint Representation	104
6.8	Timed Workflow Model Calculus	105
6.8.1	Computation of the Timed Workflow Model	106
6.9	Time Constraint Satisfiability	108
6.10	Incorporating Explicit Time Constraints	109
6.10.1	Conditional Execution Paths	110
6.10.2	Calculation/Incorporation Procedure	112
6.10.3	Lower-Bound Constraint	114
6.10.4	Upper-Bound Constraints	117
6.10.5	Valid Workflow Executions	118
6.11	Example for Computing the TWG	119
6.12	Summary	121
7	Prototypical Implementation	123
7.1	Introduction	123
7.1.1	<i>GWfD</i> Architecture	124
7.2	<i>GWfD</i> Functionalities	125
7.3	The Workflow Designer Tool <i>GWfD</i>	126
7.3.1	The Main Window	126
7.3.1.1	The File Menu	126
7.3.1.2	The Load/Save Window	127
7.3.1.3	The Extra Menu	128
7.3.2	The Specification Editing Window	128
7.3.3	The Specification and Model Window	130
7.3.3.1	Zooming	131
7.3.3.2	Calculating Time Values	132
7.3.4	Time Constraint List	132
7.3.4.1	Defining a Relational Time Constraint	133
7.3.4.2	Checking Time Constraints	134
7.3.5	The Child-Model Window	134
7.3.5.1	Applying Transformations	135
7.4	Summary	136

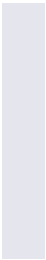
8	Conclusions and Further Work	139
8.1	Conclusions	139
8.2	Ongoing Work/Further Work	141
8.2.1	Schedules	141
8.2.2	Generalizing to Temporal Reasoning with Multiple Granularities	142
8.2.3	Relaxing the Restrictive Workflow Structure	142
	Bibliography	143
	Curriculum Vitae	157
	Electronic Data	159
	Index	161

List of Figures

2.1	Workflow System Characteristics acc. to [Hollingsworth, 1995] . . .	9
2.2	Workflow Reference Model acc. to [Hollingsworth, 1995]	10
2.3	Workflow Classification acc. to [Alonso et al., 1997]	11
3.1	Example Petri-Net acc. to [van der Aalst, 1998]	17
3.2	Workflow specification example (control structure)	20
3.3	Graphical elements	21
3.4	Workflow specification graph example	22
3.5	Sequential Execution Example	23
3.6	Parallel Execution Example	24
3.7	Conditional Execution Example	24
3.8	Alternative Execution Example	24
3.9	Iteration AE as Complex Activity	25
3.10	Iteration Split into Sequences and Conditionals	25
3.11	Workflow Model Control Structure Example	26
3.12	Workflow Model Graph Example	27
4.1	Workflow Metamodel	33
4.2	Example Workflow	44
5.1	Workflow instance types in text-based notation	49
5.2	Workflow instance types in graph-based notation	50
5.3	Workflow Example with Instance Types - 1	52
5.4	Workflow Example with Instance Types - 2	53
5.5	Flatten	55
5.6	Occurrence Encapsulation	56
5.7	Join Moving Over Activity	57
5.8	Join Moving Over Seq-Join	58
5.9	Or-Join Moving Over Or-Join	59
5.10	Alt-Join Moving Over Alt-Join	61
5.11	Or-Join Moving Over Alt-Join	62

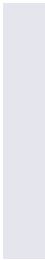
5.12	Alt-Join Moving Over Or-Join	64
5.13	Join Coalescing	65
5.14	Separating a Conditional/Alternative Path	67
5.15	Join Moving Over And-Join (Unfold) - 1	69
5.16	Join Moving Over And-Join (Unfold) - 2	70
5.17	And-Join Moving Over Or-Join	71
5.18	Split Moving Before Activity	73
5.19	Split Moving Over Seq-Split	74
5.20	Aggregated Workflow	76
5.21	Prepared Workflow	77
5.22	Unfolded Workflow	78
5.23	Intermediate Partially Unfolded Workflow	79
5.24	Partially Unfolded Workflow	80
5.25	Unfold Workflow Procedure	81
5.26	Unfold Workflow Procedure - 1	81
5.27	Unfold Workflow Procedure - 2	81
5.28	Unfold Workflow Procedure - 3	82
6.1	Process Model for Time Management in WfMSs	91
6.2	Examples of Correct and Incorrect Constraints	96
6.3	Conversion of Fixed-Date Constraints	98
6.4	Timed Workflow Metamodel with Time Components	100
6.5	Example PERT Chart	102
6.6	Gantt Chart Example	103
6.7	Example ePERT Chart acc. to [Pozewaunig et al., 1997]	103
6.8	An Occurrence Node in a Timed Workflow Graph	104
6.9	Example Timed Workflow Graph	105
6.10	Example timed workflow graph without upper bound constraints	108
6.11	Workflow Example with Conditional Execution Paths	110
6.12	Workflow Instance Types of Figure 6.11	111
6.13	Path with $ubc(B, D, 20)$ and $ubc(C, G, 15)$	112
6.14	Partially Unfolded Workflow of Figure 6.9	113
6.15	Example Workflow - Transformation Step 1 and 2	119
6.16	Example Workflow - Transformation Step 3 and 4	120
7.1	GWfD Architecture	124
7.2	Main Window of the GWfD	126
7.3	File Menu of GWfD	127
7.4	Load/Save Window	127
7.5	Database Location Options	128
7.6	Control Element Background Color Options	128

7.7	Specification Editing Window (loaded Workflow)	129
7.8	Specification and Model Window	130
7.9	Model Window with Context Menu and visible E/L-Values	131
7.10	Time Constraint List and Activated Context Menu	132
7.11	Checking Time Constraints	133
7.12	Child-Model Window of the <i>GWfD</i>	134
7.13	Partial Unfolded Workflow Model	135
7.14	Join Moving over Activity	136
7.15	Result of Join Moving over Activity of Fig. 7.14	137



List of Tables

4.1	Integrity Constraints (Brief Description)	37
5.1	Overview of Basic Workflow Transformations	54
5.2	Overview of Complex Workflow Transformations	75
6.1	Calculation instructions for timed workflow model without explicit time constraints	107



List of Definitions

3.1	Workflow	15
3.2	Structured Workflow	16
5.1	Equivalent workflows	51
5.2	Equivalent workflow instance types	51
5.3	Equivalent occurrences	51

List of Algorithms

5.1	Flatten	56
5.2	Join moving over activity	58
5.3	Or-Join Moving Over Or-Join	60
5.4	Or-Join Moving Over Alt-Join	63
5.5	Join Coalescing	66
5.6	Separating a Conditional/Alternative Path	67
5.7	Join Moving Over And-Join	68
5.8	And-Join Moving Over Or-Join	72
5.9	Split Moving Before Activity	74
5.10	Backward Unfolding Procedure	76
5.11	Partial Backward Unfold	80
5.12	Forward Unfolding Procedure	83
6.1	Checking Constraint Correctness	95
6.2	Forward Calculation	115
6.3	Backward Calculation	117
6.4	Incorporation of Upper Bound Constraints	118

1

Introduction

In today's dynamic and turbulent business environment, there is a strong need for organizations to become globally competitive. The survival key to competitiveness is (i) to be closer to the customer and deliver value added products and services within the shortest possible time and (ii) to make decisions quickly ahead of competitors. Consequently, information and information technology have become the critical strategic resource for any type of business. It results in the growing need of employees in all organization levels to efficiently and effectively use information systems to consolidate their information resources, which demands integration of business processes of an enterprise.

The solution for a business is to invest and operate with a valuable asset called *enterprise resource planning* (ERP). Enterprise resource planning use integrated cross functional software to re-engineer, integrate, and automate the basic business process of a company to improve its efficiency, agility, and profitability. In a business setting, enterprise resource planning attempts to mesh the suppliers and customers with the manufacturing environment of the organization.

Certain examples of ERP used are marketing information systems, computer-integrated manufacturing, computer-aided manufacturing, manufacturing execution systems, computer-aided engineering, computer-aided design, human resource information systems, accounting information systems, and financial management systems. These examples listed are not the only enterprise resource planning systems used; in fact, planning systems are available that are even more specific in getting the job done.

Enterprise resource planning involves many advantages, among them location is one of the positives that is fulfilled by using enterprise resource planning. It is assumed that companies implementing enterprise resource planning solutions have multiple locations of operations and control. Hence, online data transfer has to be done via locations. To facilitate these transactions, the other important en-

abling technologies for enterprise resource planning systems are Workflow (WF), Workgroup, Groupware, Electronic Data Interchange, Internet, Intranet, and Data Warehousing.

In this thesis we focus on workflows and workflow management systems (WfMSs). WfMSs improve business processes by automating tasks, getting the right information to the right place for a specific job function, and integrating information in the enterprise [WfMC, 1999b; Lawrence, 1997; Georgakopoulos et al., 1995; Hollingsworth, 1995].

1.1 Motivation for Research on Time Management

As mentioned above, one of the most critical needs in companies striving to become more competitive is the ability to control the flow of information and work throughout the enterprise in a timely manner [Eder and Panagos, 2000; Eder et al., 2000].

However, on the one hand, existing WfMSs [InC; Leymann and Roller; SAP; Ultimus] offer limited support for modeling and managing time constraints associated with processes and their activities [Bettini et al., 2002; Pozewaunig et al., 1997; Jasper and Zukunft, 1996]. This support appears mainly by means of monitoring activity deadlines [Schmidt, 1996]. Indeed, the consistency of these deadlines and the side effects of missing some of them are not addressed [Eder et al., 2000]. On the other hand, time management has only recently attracted substantial attention in the workflow research community [Marjanovic and Orlowska; Bettini et al.; Casati et al.; Dadam et al.; Bussler]. They deal with time management in a different manner each of them having their strengths and weaknesses. However, the consistency of explicit time constraints and the side effects of missing some of them are not addressed. Consequently, time-related restrictions, such as bound execution durations and/or absolute deadlines, are often associated with process activities and sub-processes. Arbitrary time restrictions and unexpected delays could lead to time violations. Typically, time violations increase the cost of business processes because they require some type of exception handling [Panagos and Rabinovich, 1997b].

Therefore, the comprehensive treatment of time and time constraints is crucial to designing and managing business processes. It is imperative that current and future WfMSs provide the necessary information about a process, its time restrictions, and its actual time requirements to process modelers and managers. The need for time management in workflows can be illustrated by the following points [Eder and Panagos, 2000]:

- At build-time, when workflow schemes are defined and developed, workflow modelers need means to represent time-related aspects of business processes (activity durations, time constraints between activities, *etc.*) and to check their feasibility.
- At run-time, when workflow instances are instantiated and their executions are started, process managers should be able to adjust time plans (e.g. extend deadlines) according to time constraints and unexpected delays. Furthermore, they need pro-active mechanisms to be notified about possible time constraint violations. Only then they can take the necessary steps to avoid time failures. Workflow participants need information about urgencies of the tasks assigned to them to manage their personal work lists. If a time constraint is violated, the WfMS system should be able to trigger exception handling to regain a consistent state of the workflow instance. Business process re-engineers need information about the actual time consumption of workflow executions to improve business processes.
- Finally, controllers and quality managers need information about activity start times and execution durations, which are usually provided by workflow systems via workflow documentation (also referred to as *workflow history* or *workflow logging*) and monitoring interfaces.

1.2 Objectives and Contributions of this Thesis

In this thesis, we are mainly interested in build-time aspects. Therefore, a new framework for time management in workflow systems is defined. In particular, we address the following issues.

Developing an advanced workflow metamodel

We present a workflow metamodel for capturing structured and less structured workflows. This metamodel supports hierarchical composition of complex activities. Both elementary and complex activities can be used in several workflow definitions and in several definitions of complex activities (re-use).

Modeling of time and time constraints to capture the available time information

We developed a control structure based workflow model which can be represented either as programming language style text or as structured graph. Each activity at the specification level is augmented with a deterministic discrete duration value and each occurrence at the model level with eight time values, which represent time points for the end event of the occurrence. Time constraints are relations between two activities of a certain type. Such constraints are depicted in tabular form or as directed edge (this is only possible in the graph based model).

Pro-active time calculations to capture time constraint violations and raise alerts in case of potential future time violations

Time calculations are required for computing optimistic and pessimistic start and finish times of activities within processes, available slack time for activities, updating existing deadlines, and so on. Typically, the assignment of external deadlines is an iterative process. We outline a technique that can be used to verify time constraint satisfiability, *i.e.*, it is possible to find a workflow execution that satisfies all constraints. If external deadlines cannot be met, the designer might modify the workflow structure, or change the deadlines.

Performing workflow transformations in order to tackle the problem of unnecessary rejections induced by superfluous time constraint violations

We consider transformations which do not change the semantics of the workflow. Such equivalence transformations are frequently needed for workflow improvements, workflow evolution, organizational changes, and for time management in workflow systems. For time management, for instance, we apply transformation operations on the process model in order to expand the model or to separate the intrinsic instance types in the process model.

Proof-of-concepts (PoC) with a prototype

We have implemented a prototype called *Graphical WF-Designer* for our time management framework. The purposes of the prototype are (i) concept exploration (proof-of-concept), (ii) demonstration, and (iii) validation.

1.3 Structure of the Thesis

Chapter 2 gives a high-level overview of the current workflow management methodologies.

Chapter 3 represents the workflow model we assume in this thesis. We describe the workflow objects and workflow control structures used in this thesis. In addition, our graphical representation of workflows is shown and illustrated by means of examples.

Chapter 4 points out several requirements for a workflow metamodel. Then the metamodel for workflow definitions that supports control structure oriented as well as graph based representation of processes is developed. Important aspects of this metamodel, such as the elaborated hierarchical composition which supports the re-use of activity definitions and the separation of specification and model level in the workflow description are demonstrated. Moreover, the necessary metamodel constraints are stated.

Chapter 5's main contribution is the development of a set of basic workflow scheme transformations and, based on them several complex transformation operations are defined. All transformation operations presented maintain the semantics of the underlying workflows. In order to accomplish equivalence-preserving transformations, we will introduce a new equivalence criterion on workflows.

Chapter 6 represents our technique for modeling, checking, and enforcing temporal constraints in workflow processes consisting of the above introduced control structures, in particular conditional and alternative control structures. We demonstrate our technique for calculating the timed workflow graph and for incorporating lower and upper bound constraints. Furthermore, we show how our technique avoids superfluous time constraint violations by means of applying transformation operations.

Chapter 7 describes the graphical workflow designer prototype that has been developed based on the introduced theoretical concepts. The workflow designer and the time management functions provided by this tool are described and illustrated with an example.

Chapter 8 summarizes the work presented in this thesis, expounds main conclusions and identifies possible areas of future work.

2

Workflow Systems Overview

This Chapter provides a general introduction to the concepts of workflow systems and technology. There is a lot of work on workflow technology and systems dealt with literature. The *Workflow Management Coalition (WfMC)* [Hollingsworth, 1995] has developed an overall standard for workflow systems. This standard describes the *Workflow Reference Model*, which identifies the characteristics, terminology, and components of *Workflow Management Systems (WfMSs)*. [Georgakopoulos et al., 1995] gives a valuable survey about workflow management technology. Moreover, there are some books that discuss workflow management technology, e.g. extracts by [Jablonski and Bussler, 1996; van der Aalst and van Hee, 2002; WfMC, 2002 2001 2000].

In the following, a brief overview of recent technologies is given. For detailed information the above cited documentation might be consulted.

2.1 Introduction

Business enterprises must deal with global competition, reduce costs, and rapidly develop new services and products. In order to cope with these requirements enterprises must permanently reconsider and optimize the way they do business and adapt their information systems and applications to support evolving business processes accordingly [Georgakopoulos et al., 1995].

In the early days, work was passed from one participant (or worker) to another. The main benefits were that work was delivered to people, and each worker could assume that work was ready for processing, since the workflow system would not forward incomplete items. Then delivery was automated. Consequently, workflow technology matured and the process of delivery was automated. A work item or data set was created, and it was processed and changed in stages at a number of

processing points to meet business goals [WfMC, 2001]. Workflow technology facilitates this by providing methodologies and software to support:

- business process modeling to capture business processes as workflow specifications.
- business process re-engineering to optimize specified processes.
- workflow automation, which consists of
 - process instantiation and control, and
 - interaction with users and applications.

During the last few years tools have been developed to not only do the work, but to manage the workflow. Such a workflow is represented as a workflow process, which is defined in a form that supports automated manipulation in the workflow computer system. The process is managed by a computer program that assigns the work, passes it on, and tracks its progress. The workflow process is traditionally defined in office terms, i.e. moving the paper, processing the order, issuing the invoice. But the same principles and tools apply to filling the order from the warehouse, assembling documents, parts, tools, and to people repairing a complex system, or manufacturing the complex device [WfMC, 2001].

An effort to provide a consensus *Workflow Reference Model* [Hollingsworth, 1995] was made by the WfMC, gathering ideas from researchers in the field. The WfMC published a glossary [WfMC, 1999a 1994ba] of useful terms related to workflow. This glossary defines workflow as:

The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Workflow normally comprises a number of logical steps, each of which is known as an activity. Thus, an activity can involve manual interaction with a user or workflow participant, or it might be executed using machine resources [WfMC, 2000].

2.1.1 Workflow Management System

A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines. The WfMS is able to interpret the process definition, interact with workflow participants and, if required, to invoke the use of IT tools and applications [WfMC, 2000].

All WfMSs may be characterized as providing support in three functional areas (see Figure 2.1):

- The build-time functions, concerned with defining the workflow process and its constituent activities.
- The run-time control functions concerned with managing the workflow processes in an operational environment and sequencing the various activities.
- The run-time interactions with human users and IT application tools for processing the various activity steps.

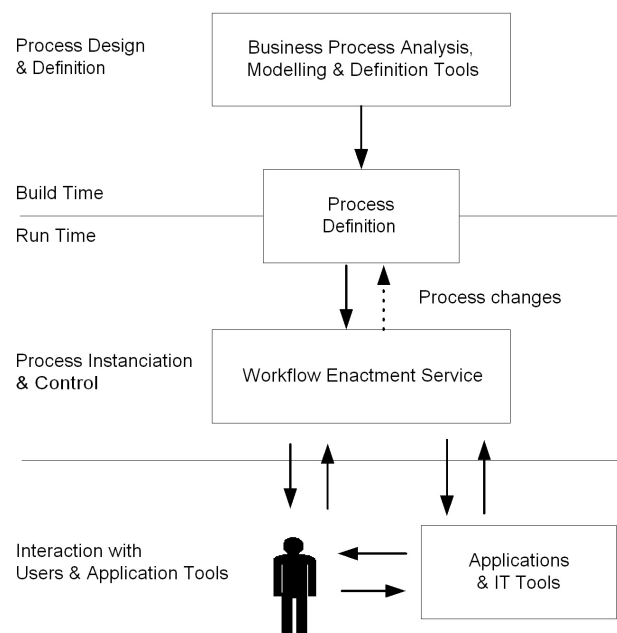


Figure 2.1: Workflow System Characteristics acc. to [Hollingsworth, 1995]

2.1.2 Workflow Reference Model

The Workflow Management Coalition has developed an overall framework for the establishment of workflow standards. This framework includes the *Workflow Reference Model* (see Figure 2.2) that has been developed from the generic workflow application structure by identifying the interfaces within this structure.

All workflow systems contain a number of generic components which interact in a defined way. To achieve interoperability between workflow products a standardized set of interfaces and data interchange formats between such components is necessary. Therefore, the WfMC defined an interoperability protocol for each interface within the workflow architecture as illustrated in Figure 2.2. The

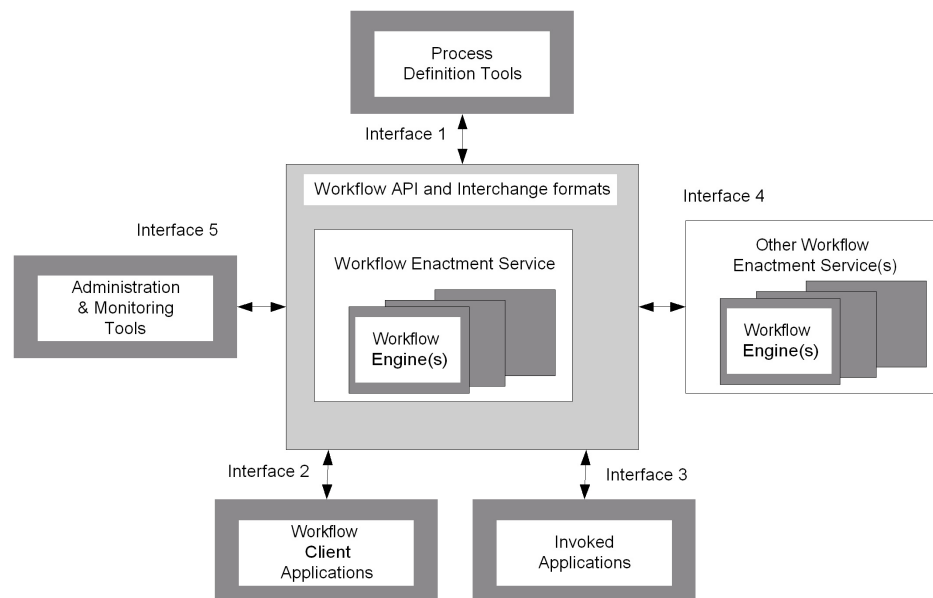


Figure 2.2: Workflow Reference Model acc. to [Hollingsworth, 1995]

architecture of the Workflow Reference Model identifies five interfaces. These interfaces are related to the workflow enactment service, and they are supported by a set of API calls (WAPI). A detailed description can be found in [Hollingsworth, 1995].

2.1.3 Workflow Classification

A widely accepted workflow classification distinguishes between *administrative*, *ad hoc*, *collaborative*, and *production* workflows. The parameters of this classification are the execution frequency of the business processes and their value to the associated enterprises. However, it is also possible to organize them according to the task complexity and the task structure [Alonso et al., 1997; Georgakopoulos et al., 1995]. Figure 2.3 illustrates these classifications.

These different classes are characterized as follows [Alonso et al., 1997; Georgakopoulos et al., 1995]:

- **Administrative workflows** refer to bureaucratic processes where the steps to follow are well established. There is a set of rules known by everyone involved. Examples are the registration for courses in a university, registration of a vehicle, and almost any other process in which there is a set of forms to be filled in and routed through a series of steps.
- **Ad Hoc workflows** are similar to administrative workflows except for the fact that they tend to be created to deal with exceptions or unique situa-

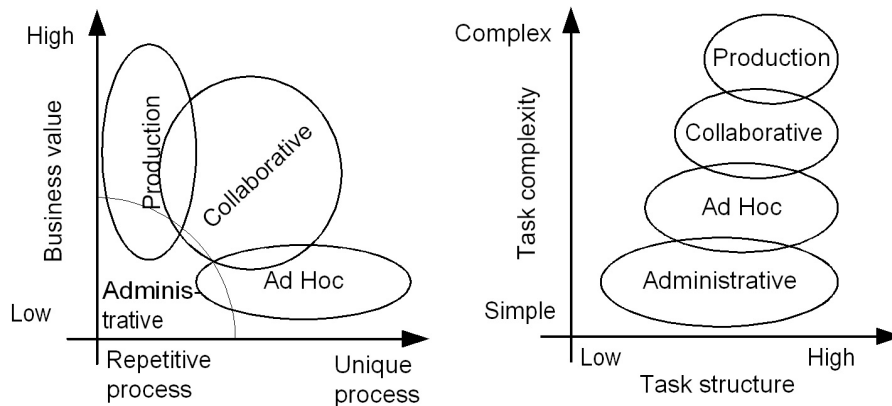


Figure 2.3: Workflow Classification acc. to [Alonso et al., 1997]

tions. This depends on the users involved. While for a university the process of applying for a degree is an administrative procedure, for a student it is something that happens only once and therefore ad hoc. If the process is of sufficient complexity, it is possible to define a workflow to help with its coordination and management. It may also be the case that the situation is not exceptional, but each particular instance is unique.

- **Collaborative workflows** are mainly characterized by the number of participants involved and their interactions. Unlike other types of workflows, which are based on the premise that there is always forward progress, a collaborative workflow may involve several iterations over the same step until some form of agreement is reached; or it may even involve the steps back to an earlier stage. The writing of a paper by several authors would serve as a good example. Additionally, it must be pointed out that collaborative workflows tend to be very dynamic in the sense that they are defined as they progress.
- **Production workflows** can be characterized as repetitive and predictable business processes, which are directly related to the function of the organization. Credit and loan applications and insurance claims are typical examples, but the difference between administrative and production workflows is sometimes a matter of perspective. Usually, when talking about production workflows, the main points to consider are the large scale, the complexity and heterogeneity of the environment where they are executed, the variety of people and organizations involved, and the nature of the tasks.

Another classification often found in the literature is corresponding the underlying technology, mail-centric, document-centric and process-centric. Mail-centric systems are based on electronic mail and can be roughly associated with collaborative and ad hoc workflows. Document-centric systems are based on the idea of routing documents and the ability to interact with external applications is limited. Administrative workflows based on forms can be implemented using document-centered systems. Process-based systems correspond to production workflows. They generally implement their own communication mechanisms, they are built on top of databases and provide a wide range of interfaces to allow interaction with legacy and new applications [Alonso et al., 1997].

2.2 Summary

This Chapter provides a brief overview of workflow technology necessary for this thesis. Starting out with the workflow definition it gives an introduction to the notion of workflow management systems and its components. Finally, the workflow reference model and a widely accepted workflow classification is mentioned.

3

Workflow Modeling

Workflow modeling is an effective technique for understanding and documenting existing business processes. Support for heterogeneous processes (human-centered and system-centered), flexibility and reuse are important challenges for the design of process modeling languages (cf. [van der Aalst and van Hee, 2002; van der Aalst et al., 2002; Joeris and Herzog, 1999]). Flexible and collaborative processes require human intervention. Therefore, a process modeling language which on a high level of abstraction is needed is easy to use and supports the visualization of its elements. Based on these design objectives and requirements, we have developed a process modeling language which supports flexible workflow specifications.

In this Chapter, we concentrate on workflow modeling concepts for heterogeneous processes under the given design goals. On the conceptual level, we focus on the specification of complex user-defined control flow dependencies which can be used at a high level of abstraction and which are reusable in different contexts.

This Chapter is organized as follows: in Section 3.1 the notion of workflow modeling is introduced. Section 3.2 discusses related work in literature, and Section 3.3 provides definitions for technical terms used in this Chapter. Section 3.4 gives an overview of common process modeling approaches. Section 3.5 shows our corresponding graphical model for workflows. Section 3.6 explains the control structures appearing in our specifications, and Section 3.7 describes our composite workflow model. Finally, Section 3.8 compiles conclusions drawn from the above Sections.

3.1 Introduction

In order to build a workflow specification, it is necessary to start with basic modeling and representation concepts.

Numerous workflow models have been developed based on different modeling concepts (e.g. Petri Net variants, precedence graph models, precedence graphs with control nodes, state charts, control structure based models, etc.) and on different representation models (programming language style text based models, simple graphical flow models, structured graphs, etc.) [Eder and Gruber, 2002]. With respect to different points of view each of the listed modeling and representation concepts has its characteristics, strengths and weaknesses. Therefore, a workflow described in a particular model may be more suitable for a specific kind of consideration (e.g. conceptual comprehension) than in a different model. Consequently it becomes necessary that a workflow described in a particular model is transformed to a different model to inspect it from a different point of view.

Transformations between representations can be difficult (e.g. the graphical design tools for the control structure oriented workflow definition language *WDL* of the workflow system *Panta Rhei* had to be based on graph grammars to ensure expressiveness equality between text based and graphical notation [Eder et al., 1997b]).

In this Chapter, the main contribution is the presentation of a workflow model for control structure oriented as well as graph based processes. Important aspects are the elaborated hierarchical composition supporting re-use of activities by means of *occurrences*.

In the following, an overview of different concepts as well as a detailed description of the concepts used in this thesis is given.

3.2 Related Work

Many languages have been proposed for the specification of workflow processes. Some of these languages are based on existing modeling techniques, such as precedence graphs, Petri nets and State charts. Other languages are system specific. Any attempt to give a complete overview of these languages is destined to fail [van der Aalst et al., 2002]. Throughout this Chapter we will refer to common languages without striving for completeness.

3.3 Workflow Definition

We specify a more sophisticated definition of workflows as it is done in Section 2.1.

DEFINITION: A workflow is a collection of *activities*, *agents*, and *dependencies* between activities. Activities correspond to individual steps in a business process, agents (software systems or humans) are responsible for the enactment of activities, and dependencies determine the execution sequence of activities and the data flow between them [Eder and Gruber, 2002; Eder et al., 2000]. (3.1)

3.3.1 Conformance Classes

According the WfMC ([WfMC, 1999b]) there are three conformance classes restricting the transitions (dependencies) between activities. The conformance classes are defined by the WfMC as follows:

- **non-blocked** There is no restriction for this class.
- **loop-blocked** The activities and transitions of a process definition form an acyclic graph.
- **full-blocked** For each join there is exactly one corresponding split of the same kind and vice versa. (for the notion of join and split see Section 3.5.2 or 4.4).

Workflows complying with the first two classes may lead to structural conflicts such as deadlocks and lack of synchronization [Sadiq and Orlowska, 1999a; Lin et al., 2002]. Therefore, these workflows have to be verified to ensure their correct execution, whereas workflows of the full-blocked class are per se structurally conflict-free. Workflows of the third conformance class are also called (*well*) *structured workflows* [Eder and Gruber, 2002] in this thesis, and they can be defined as it is stated in the subsequent Section. In [Kiepuszewski et al., 1999], a similar recursive definition of structured workflows is denoted. Although structured workflow models are less expressive than arbitrary workflow models [Kiepuszewski et al., 1999], the concepts of time management can be shown more easily on structured workflow models than on non-blocked models.

3.3.2 Structured Workflow Definition

At present we assume that workflows are *well structured*.

DEFINITION: A well-structured workflow consists of m sequential activities, $T_1 \dots T_m$. Each activity T_i is either elementary, i.e. it cannot be decomposed any further, or it is complex. A complex activity consists of n_i parallel, sequential, conditional or alternative sub-activities $T_i^1, \dots, T_i^{n_i}$, each of which is either elementary or complex (cf. [Eder and Gruber, 2002]). (3.2)

Typically, well structured workflows are generated by workflow languages with the usual control structures which adhere to a structured programming style (e.g. Panta Rhei [Eder et al., 1997b]).

3.3.3 Occurrence(s) of Activities

Within a complex activity a particular activity may appear several times. To distinguish between those appearances, we introduce the notion of occurrences [Eder and Liebhart, 1995; Liebhart, 1998; Eder and Gruber, 2002]. An occurrence is associated with an activity and represents the place where an activity is used in the specification of a complex activity. Each occurrence, therefore, has different predecessors and successors.

The distinction between an activity and its (multiple) occurrence(s) is important for the feature of re-useability, i.e. an activity is defined once and used several times in workflow definitions. Equally, for maintenance it is only necessary to change an activity once, and all its occurrences are changed as well. This allows new workflows to be easily composed using predefined activities. A composition of this kind can be called a *workflow specification*.

3.4 Workflow Models

In order to build a workflow specification, it is necessary to start with modeling concepts. Numerous workflow models have been developed. In this Section a brief overview of some common workflow models is given.

■ Petri Net Variants

Basically, in a Petri net-based model, activities are modelled as transitions. Arcs represent dependencies between activities, and places are used to model the internal states of the workflow. Conditional and alternative branching are modelled by using a place with multiple outgoing arcs. A transition with multiple outgoing arcs represents the parallel execution of activities. In order to avoid deadlocks, a parallel branching must end with a so-called

synchronization transition which has multiple incoming arcs [van der Aalst and van Hee, 2002; van der Aalst, 1998].

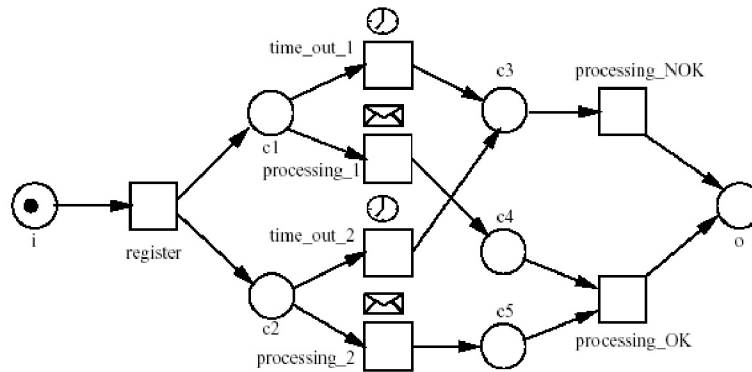


Figure 3.1: Example Petri-Net acc. to [van der Aalst, 1998]

■ Precedence Graphs

A precedence graph is a directed acyclic graph whose nodes represent tasks and whose edges indicate the order in which tasks are to be accomplished (loops have to be unfolded). A directed edge from node u to node v means that the task corresponding to u must be done before the task corresponding to v . In other words, a task is executed as soon as all its predecessors have completed. Numerous descendant methodologies like PERT and CPM are widely used, especially in project management and product development domains [Pozewaunig et al., 1997].

■ Precedence Graphs with Control Nodes

A precedence graph with control nodes is an above described precedence graph which is augmented with control nodes like split and join nodes in order to establish control structures such as parallel, conditional and alternative structures (s. Section 3.6). Generally, a split node refers to a control element having several immediate successors, which are executed depending on the control structure. A join node refers to a control element that is executed after (a) particular immediate predecessor(s) finish(es) execution. This depends on the control structure [Eder and Gruber, 2002; Zhao and Stohr, 1999].

■ State Charts

State charts were originally developed for reactive systems (e.g. embedded control systems in automobiles) and have been quite successful in this area. State charts reflect the behavior of a system in that they specify the control

flow between activities. A state chart is basically a finite state machine with a distinguished initial state and transitions driven by EventConditionAction rules (ECA rules). Each transition arc between states is annotated with an ECA triple. A transition from state X to state Y fires if the specified event E occurs and the specified condition C holds. The effect is that state X is left, state Y is entered, and the specified action A is executed. Conditions and actions are expressed in terms of data item variables; conditions and events may also refer to states and the modification of variables by means of special predicates like $in(s)$ and $ch(v)$ where s is a state and v is a variable. In addition, an action A can explicitly start or stop an activity and can generate an event E or set a condition C [Wodtke and Weikum, 1997].

■ Script Languages

Script languages allow the description of workflows in a textual form and their “look and feel” is similar to that of traditional programming languages. Thus the workflow definition language WDL for the system Panta Rhei ([Eder et al., 1997b]) consists of five basic units: the workflow specification part, the definition of activities, roles, organization structures and inter-process communication.

The workflow specification part of WDL enables the formal description of a workflow (process). Primarily this serves to specify the control and data flow of a workflow. Additionally, transactional and time relevant aspects are specified within this part.

The structure of a typical workflow specification is similar to the structure of ordinary procedural programs:

- ▶ **Header:** Every workflow has a name and any number of optional arguments. Arguments are forms which are passed into the workflow and/or produced by the workflow.
- ▶ **Declaration Part:** The declaration part consists of two units: workflow declaration and data declaration. The workflow declaration part serves the specification of general process information. This type of information is equal for all processes and comprises information of the process owner, a subject text, a short description of the current process, the specification of the maximal execution time and the specification of some action (e.g. the execution of a special time activity) in case of a time failure. Forms which are produced within a workflow have to be declared in the data declaration part. This is simply done by assigning form types to form variables. Fields within the form can be accessed via the dot notation. The scope of a data declaration is the whole workflow.

- **Body:** This is the main part of a workflow specification. The description of the control and data flow of the process is expressed between the keywords *begin* and *end*. Within the workflow context this means to answer the following W-questions: *who has to do what and when and with which data*. Based on this paradigm, typical statement sequences in WDL are of the following simple structure:

```
agent name activity name (form names);
```

The semantics of this is: The workflow agent `agent name` performs the activity `activity name` with the forms defined by the arguments `form names`.

WDL offers a variety of control structures to specify the execution order of workflow steps. The most relevant control structures are sequence, alternative and parallelism (and-parallelism and or-parallelism).

■ Rule-based Languages

In rule-based languages workflows are composed by a set of rules. Each rule is composed by two parts, which are called left-hand side (*lhs*) and right-hand side (*rhs*), e.g. by $lhs \Rightarrow rhs$. The program executes on the content of the working context. The effect of a computation is the successive application of the rules in the program to the content of the working context. The effect of applying a rule to the working context is either the removal of a record or the introduction of a new record. The *rhs* of the rule contains the action used to modify the content of the working context. The *lhs* of each rule represents the conditions that have to be met for the rule to be applicable. The *lhs* is composed by a sequence of conditions, where each condition is a pattern. A pattern is a condition on the content of the working context. The pattern describes a record by specifying the values for certain fields. The pattern is satisfied if there is one record in the working context that meets the requirement specified by the pattern. The programmer is allowed to introduce to the program a number of initial rules that are going to be executed first in the program. The assumption is that when the program starts, the initial rules are going to be executed; their effect is to add the mentioned records to the working context [Wichert et al., 2001; Bonner, 1999; Casati et al., 1996a; Kappel et al., 1995].

As the subsequent example rule `R.GetOffers` shows, the action is executed by sending the message `notifyAgent` to the instance `a_get_offers` if the condition evaluates to true.

```

DEFINE RULE R.GetOffers
  ON POST (Activity, perform: aFolder) DO
    IF Offers notUpToDateFor: ((aFolder at: 'Order') positions)
    THEN
      EXECUTE a_get_offers notifyAgent
    ACTIVATED FOR (a_create_order_form).

```

Example Rule acc. to [Kappel et al., 1995]

3.5 Workflow Representation

A workflow specification is also based on representation concepts. In literature different representation models such as programming language style text based models, simple graphical flow models, structured graphs, etc. were introduced. In this Section an overview of our workflow representation models is given.

3.5.1 Programming Language Style Text Based Representation

Usually, well-structured workflows are generated by workflow languages with the usual control structures which adhere to a structured programming style (e.g. Panta Rhei [Eder et al., 1997b]).

WFS1:WF-Spec	A1:Activity	A2:Activity	A3,A4,A5:Activity
sequence	sequence	conditional	elementary
O1:A1	O2:A2	O5:A4	end
end	O3:A3	O6:A5	
end	O4:A2	end	
	end	end	
	end		

Figure 3.2: Workflow specification example (control structure)

Figure 3.2 shows an example of a script language model based workflow. The control structures define complex activities. Here, the workflow specification for workflow *WFS1* represents a sequence of occurrence *O1*, where the occurrence *O1* stands for the activity *A1*. Activity *A1* is a complex activity; it consists of three sequential occurrences, namely *O2*, *O3*, and *O4*, where they embody the activities *A2*, *A3*, and again *A2*. *O2* is the predecessor of *O3*, and *O3* is the predecessor of *O4*, which is implicitly expressed by the ordering of the occurrences. The complex activity *A2* represents a conditional structure with the occurrences *O5* and *O6*. Occurrences *O5* and *O6* stand for activities *A4* and *A5*. Activities *A3*, *A4* and *A5* are elementary activities.

3.5.2 Workflow Graphs

Structured workflows can also be represented by *structured workflow graphs* (SWG), where nodes represent activities or control elements and edges correspond to dependencies between nodes. An *and-split* node refers to a control element having several immediate successors, all of which are executed in parallel. An *and-join* node refers to a control element that is executed after all of its immediate predecessors finish execution. An *or-split* node refers to a control element whose immediate successor is determined by evaluating a boolean expression (conditional) or by choice (alternative). An *or-join* node refers to a control element that joins all the branches after an or-split.

The graph representation of a workflow definition can be structured in a similar way such as it is the case in the text-based representation (block-structured workflow languages) of the workflow definition. Directed edges stand for dependencies, hierarchical relations, and part-of relations between nodes. Figure 3.3 shows the graphical elements.

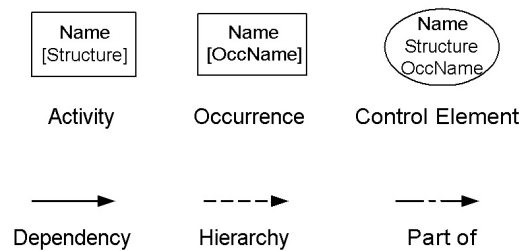


Figure 3.3: Graphical elements

Activity and occurrence nodes are represented by a rectangle in which the name of the representing activity or occurrence is indicated. An activity node additionally features the structure of the representing activity (only in complex activities). The *structure* indicates the control structure ('seq', 'par', 'cond' or 'alt') of a complex activity. At the model level (s. Section 3.7), nodes feature the name of the related specification occurrence that is encapsulated between round brackets. Control elements are represented by a circle in the graph in which the name and the structure of the control element is indicated. Furthermore, any existing predicate of a node will be depicted between angle brackets below the graphical element. Directed edges stand for dependencies, hierarchical relations (dashed line directed edge), and part-of relations (dot and dash line directed edge) between nodes.

Figure 3.4 shows the workflow defined in Fig. 3.2 in graph notation. In the graphical representation, (implicit) relationships in the text-based specification

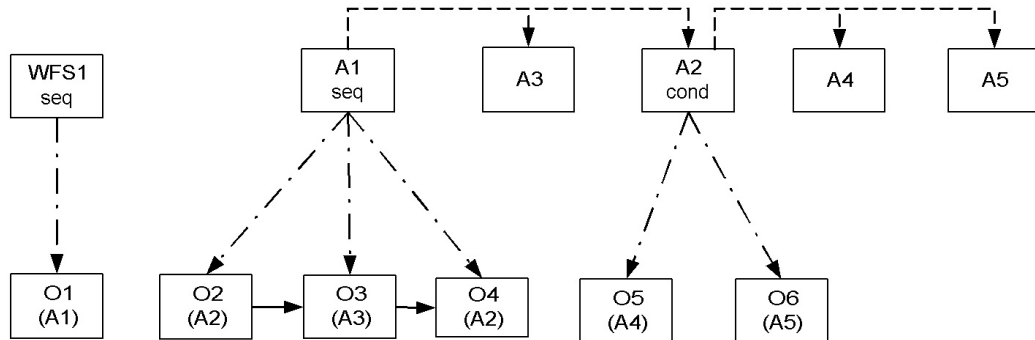


Figure 3.4: Workflow specification graph example

are explicitly expressed by the different directed edges (*dependency*, *hierarchy*, and *part-of*).

Regardless of the structured workflow definition, a structured workflow graph has two different emergences as it is described below.

3.5.2.1 Strictly Structured Workflow Graph

A workflow graph is *strictly structured* if each split node is associated with exactly one join node and vice versa, and each path in the workflow graph originating in a split node leads to its corresponding join node [Eder and Gruber, 2002].

3.5.2.2 Less Strictly Structured Workflow Graph

To allow more transformations (see Section 5) and the separation of workflow instance types in the workflow model we also offer a less strict notion. Here, a split node may be associated with several join nodes, however, a join node corresponds to exactly one split node. Each path originating in a split node has to lead to an associated join node. Such graphs result from equivalence transformations necessary e.g. for time management [Eder and Gruber, 2002].

Both representations of workflows can be freely mixed in our approach which we call *hybrid graph* or *graph-based* workflow representations. Graph based structures can be used as complex activities, and structured composite activities in turn can be used in graph based representations.

3.6 Workflow Control Structures

In this Section control structures capturing elementary aspects of process control are discussed. A workflow model is created by composition of these control structures [van der Aalst et al., 2000; Marjanovic and Orłowska, 1999a].

The figures below describe the control structures in two representation models, namely as graph based representation and as programming language style text based representation. All of the following control structures are listed in [van der Aalst et al., 2002]. Additionally, a general notion of control structures can be found in [WfMC, 1999a].

3.6.1 Sequence

A sequence (sequential execution) is the most basic workflow pattern. It is required for a dependency between two or more activities, so that one activity can be started after the predecessor activity is finished.

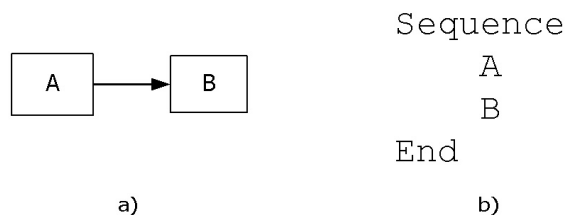


Figure 3.5: Sequential Execution Example

3.6.2 Parallelism

All activities of a parallel structure are executed. A parallel structure is necessary when two or more activities need to be executed parallel, thus allowing activities to be executed simultaneously or in any order.

3.6.3 Conditional

Exactly one of several (mutually exclusive) activities that satisfy a given condition (predicate) is executed (XOR). There is always one activity at execution time that satisfies the given condition. The activity that is executed next depends on data and state that are generated during the execution of the workflow process instance.

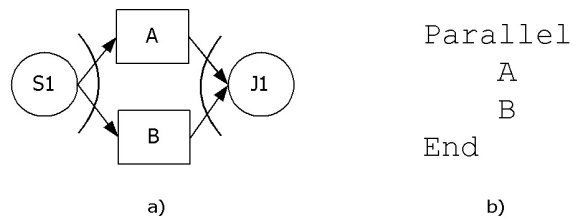


Figure 3.6: Parallel Execution Example

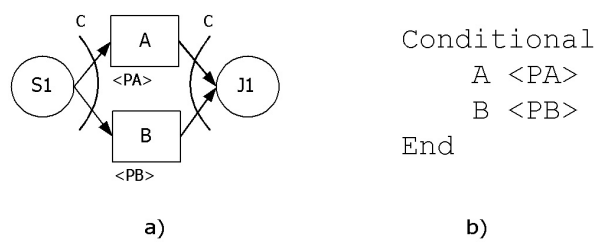


Figure 3.7: Conditional Execution Example

3.6.4 Alternative

Any activity among many alternative activities can be executed. The activity that is executed next depends on policies and information that are shared by all alternative activities of the alternative structure, and which are determined by agents. This means that any alternative will lead to a correct workflow execution [Eder et al., 1999a].

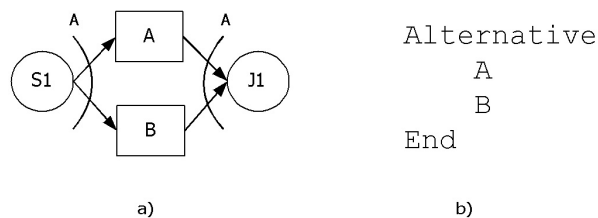


Figure 3.8: Alternative Execution Example

3.6.5 Iteration

In our approach loops of an activity are simply considered a complex activity. Alternatively, loops could be introduced as repetitions of a workflow graph. It might not be possible to exactly predict the number of iterations that will be executed during run-time. For this reason it is necessary to split the iteration into a structure that is composed of sequences and conditionals which connect the complex activity multiple times [Eder and Pichler, 2002].

In the following we assume that the workflow-graph of Figure 3.9 is embedded in the complex iteration (loop) activity *AE* that is bound by *M0* and *M5*.

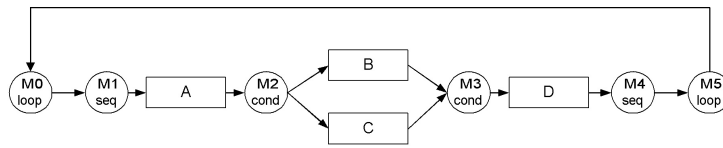


Figure 3.9: Iteration AE as Complex Activity

In the following it is necessary to split the iteration into a structure that is composed of sequences and conditionals which connect the complex activity *AE* multiple times, as shown in Figure 3.10. A detailed consideration about that conversion can be found in [Eder and Pichler, 2002].

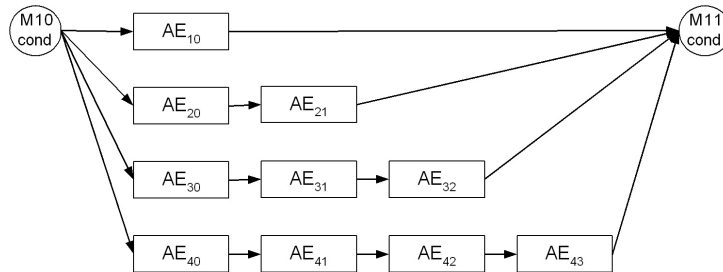


Figure 3.10: Iteration Split into Sequences and Conditionals

3.7 Composition Structure of the Workflow Model

In the workflow specification, the concept of occurrences helps to distinguish between several referrals to the same activity within a complex activity. When a complex activity is used several times within a workflow, we also have to distinguish between the different appearances of occurrences. Therefore, a model is required that corresponds to the specification, so that for the definition of the

dependencies between activities an occurrence of an activity in a workflow has to be aware of its process context. Therefore, we transform the design information (specification) contained in the metamodel into a composition, i.e. tree-like structure. In such a tree-like structure different appearances of the same activity are unambiguously distinguished, so that we can define the dependencies between activities on the basis of these occurrences. We call these items *model elements*, and the workflow consisting of model elements the *workflow model*.

WFM1:WF-Model	M1:ModelElem	M2:ModelElem	M4:ModelElem	M3,M5,M6,M7,M8:
sequence	sequence	conditional	conditional	ModelElem
M1:O1	M2:O2	M5:O5	M7:O5	elementary
end	M3:O3	M6:O6	M8:O6	end
end	M4:O4	end	end	
	end	end	end	
	end			

Figure 3.11: Workflow Model Control Structure Example

Figure 3.11 shows the workflow model for the workflow specification in Figure 3.2. In the following example, the model elements *M2* and *M4* have their own contexts with *M5* and *M6* or *M7* and *M8* respectively, and they are built up like a tree. Figure 3.12 shows the workflow model in graph notation, giving the full unflattened model in the upper half and the full flattened model in the lower half (s. Section 5.5.1.1).

3.8 Summary

This Chapter presented an overview of workflow modeling and representation concepts, emphasizing the control perspective. Some common modeling languages (text based as well as graphical) are pointed out briefly. Apart from the declaration of necessary definitions, all relevant workflow control structures are specified and described. Finally, the workflow model that is used in this thesis and which is deduced from the workflow specification closes the Chapter.

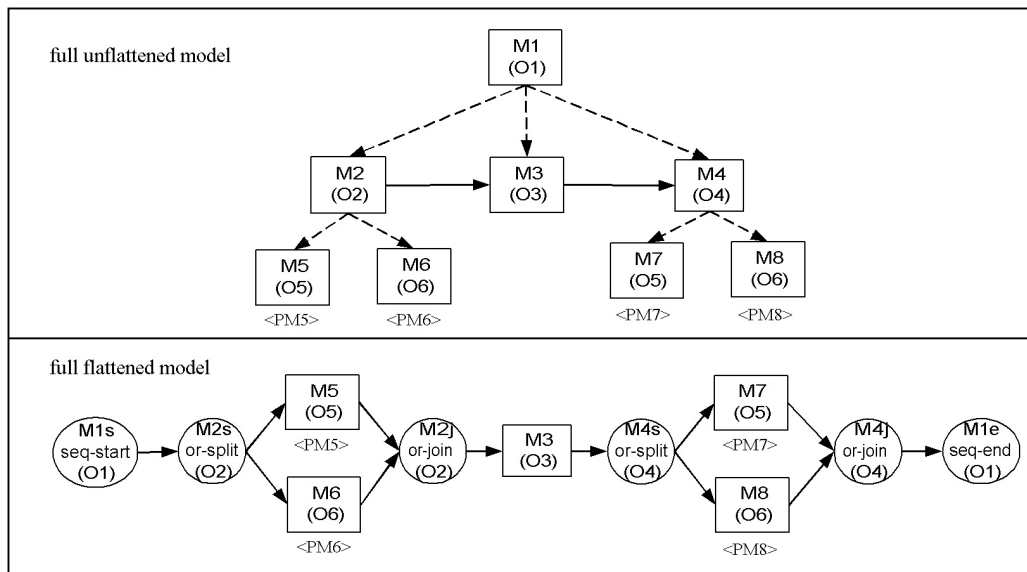


Figure 3.12: Workflow Model Graph Example

4

Workflow Metamodel

The objective of this Chapter is to introduce the metamodel that is used in this work as basis for our temporal workflow model and workflow transformations. It comprises a set of modeling concepts to capture various aspects of workflows. In particular, the workflow metamodel presented allows to model structural, functional, reuse and informational aspects of workflows (cf. [Eder and Gruber, 2002]).

This Chapter is organized as follows: Section 4.1 briefly introduces workflow metamodels, and Section 4.2 discusses several requirements for a workflow metamodel. Section 4.3 provides a brief overview of related work in literature. Section 4.4 describes in detail our workflow metamodel and Section 4.4.5 shows the semantical integrity constraints for the metamodel. Section 4.5 shows a workflow which complies with our metamodel. Section 4.6 compiles conclusions drawn from the above Sections.

4.1 Introduction

Workflow modeling is the process of creating an abstract representation of a business process. Models serve several purposes, such as testing a workflow before its application, communicating with customers, visualizing the working process, and reducing complexity etc. A particular workflow must obey predefined rules. These rules are specified in what is called a paradigm. The paradigm essentially defines the entities and relationships allowed in the given domain. The paradigm, in this sense, defines the syntax of the modeling language. The paradigm is designed to fit for particular domain, and therefore describes the way that models are used.

In general, the paradigm is represented by the specification of a metamodel. It formally defines the syntax, semantics, presentation, and translation specifications of a particular modeling environment. Here, the workflow metamodel establishes

the modeling semantics required to specify the aspects of business processes, thus it is a formalization of all available design concepts.

In this Chapter, we define a new metamodel for workflow specifications that supports control structure oriented as well as graph based representation of processes as introduced in [Eder and Gruber, 2002]. Important aspects of this metamodel are the elaborated hierarchical composition supporting re-use of activity definitions and the separation of specification and model level in the workflow description. Moreover, metamodel constraints that have to be fulfilled in order to guarantee the integrity of our workflow model are stated.

4.2 Metamodel Requirements

In this Section, several requirements for workflow metamodels are briefly discussed to sufficiently describe the structure and characteristics of a workflow by a workflow metamodel.

Generally, the aspects of workflows, which have to be definable by using a workflow metamodel, depend on the requirements resulting from the application domain. The completeness of a workflow metamodel is relative to its application so, in one application data flows do not have to be modeled, or in another one deadlines play an important role ([Eder et al., 1999b; Marjanovic and Orlowska, 1999b]). Since constructing a workflow metamodel that satisfies all requirements from all potential application domains is hardly feasible, there is no such thing as a “complete” workflow metamodel [Kradolfer, 2000]. Nonetheless, there are several common aspects of workflows that are relevant in many applications: (1) control-flow (or process) perspective, (2) resource (or organization) perspective, (3) data (or information) perspective, (4) task (or function) perspective, (5) operation (or application) perspective. Other perspectives are the causal aspect (regulations and dependencies), the historical aspect (logging), the transactional aspect (workflow consistency), etc. [Becker et al., 2000; Curtis et al., 1992; Jablonski and Bussler, 1996; Sadiq and Orlowska, 1999b]. In the following, the most important aspects are briefly described:

- **Control-flow aspect:** The control-flow perspective specifies which tasks need to be executed and in what order (i.e. the routing). The different control structures are described in Chapter 3.
- **Organizational aspect:** In the resource (organizational) perspective, the organizational structure and the population are specified. Resources ranging from humans to devices form the organizational population are mapped onto resource classes. (Which resources have to perform which activities?)

- **Informational aspect:** The data perspective deals with control and production data. Control data are data introduced solely for workflow management purposes. Control data are often used for routing decisions in conditional structures. Production data are information objects (e.g. documents, forms, and tables) whose existence does not depend on workflow management. (Which data are produced and consumed by workflows, and how do they flow?)
- **Functional aspect:** The task (functional) perspective describes the content of the process steps, i.e. it describes the characteristics of each task. A task is a logical unit of work with characteristics such as the set of operations that need to be performed, description, expected duration, due-date, priority, trigger. These applications create, read, or modify control and production data in the data perspective.
- **Operational aspect:** In the operational perspective, the elementary actions are described. Note that one task may involve several operations. These operations are often executed using applications ranging from a text editor to custom-built applications for performing complex calculations. Usually, these applications create, read, or modify control and production data in the data perspective.

Besides the aforementioned requirement aspects, there are further non-functional requirements a workflow metamodel should fulfill: (1) enactability, (2) ease of use, (3) correctness criteria, (4) evolution, and (5) reuse (see Sect. 3.3.3).

In the following, we present a metamodel that has been tailored to support workflow transformations and time management. Therefore it does not contain all necessary components (perspectives) of a workflow metamodel. In this Chapter we present a metamodel for capturing (strictly and less strictly) structured workflow models in the form of classically nested control structure representation as well as the frequently used graph representations. This metamodel supports hierarchical composition of complex activities. Both elementary and complex activities can be used in several workflow definitions and in several definitions of complex activities (re-use). A sophisticated feature of our metamodel is that it divides the workflow specification and the (expanded) workflow model(s). Moreover, our metamodel includes some specific functionalities such as transformation operations (flatten, unflatten, unfold, etc.).

4.3 Related Work

A vast number of workflow metamodels has been proposed yet (see e.g. [Casati et al., 1995; Hofstede et al., 1996; Jablonski, 1994; Joeris and Herzog, 1998; Kap-

pel et al., 1995; Kradolfer, 2000; Krishnakumar and Sheth, 1995; Liebhart, 1998; Meyer-Wegener and Böhm, 1999; Sadiq et al., 2000; Trajcevski et al., 2000]). Despite the standardization efforts taken by the WfMC [WfMC, 1999b] no generally accepted metamodel has been developed so far.

4.4 Workflow Metamodel Description

In this thesis, UML is used as meta-modeling language and OCL (Object Constraint Language) as integrity constraint language [Warmer and Kleppe, 1999].

The metamodel shown in Figure 4.1 [Eder and Gruber, 2002] gives a general description of the static scheme aspects (the build time aspects) of workflows. The metamodel presented in Fig. 4.1 is adopted to the purpose of this thesis and does therefore not contain all necessary components of a workflow metamodel. In the following, we discuss the elements of the metamodel in detail. The important concepts along with examples have already been described in the previous Chapters.

4.4.1 Metamodel Stratifications

The metamodel consists of the following parts (stratifications): the specification level contains the description of workflow types and activity types together with their composition structure via occurrences (see below). The model level contains the expanded workflow specifications, so that all activity appearances may have their individual characterizations (like due dates, agent, etc.). The instance level and the organization level are omitted in this work (for these aspects see i.e. [Eder et al., 2002]). We did not take the dimension of data into account, since there the differences between workflow systems are too big to allow a more general treatment.

4.4.2 Workflows and Activities

The class *Workflow* contains the typical header information of a workflow. It describes the process itself and has the unique attribute *wfId* for the identification, and the attributes *name* and *description*. Further administrative information, e.g. *author* and *creation date* is also available.

4.4.2.1 Activity

A workflow uses activities which are represented by the class *Activity* that has the attributes *aID* (unique), *name*, *description*, *precondition* (expressing constraints on when the activity can be called), *postcondition* (expressing the results delivered by the activity when it successfully finishes execution) and *duration*. Activities are either (external) workflows, elementary activities or complex activities

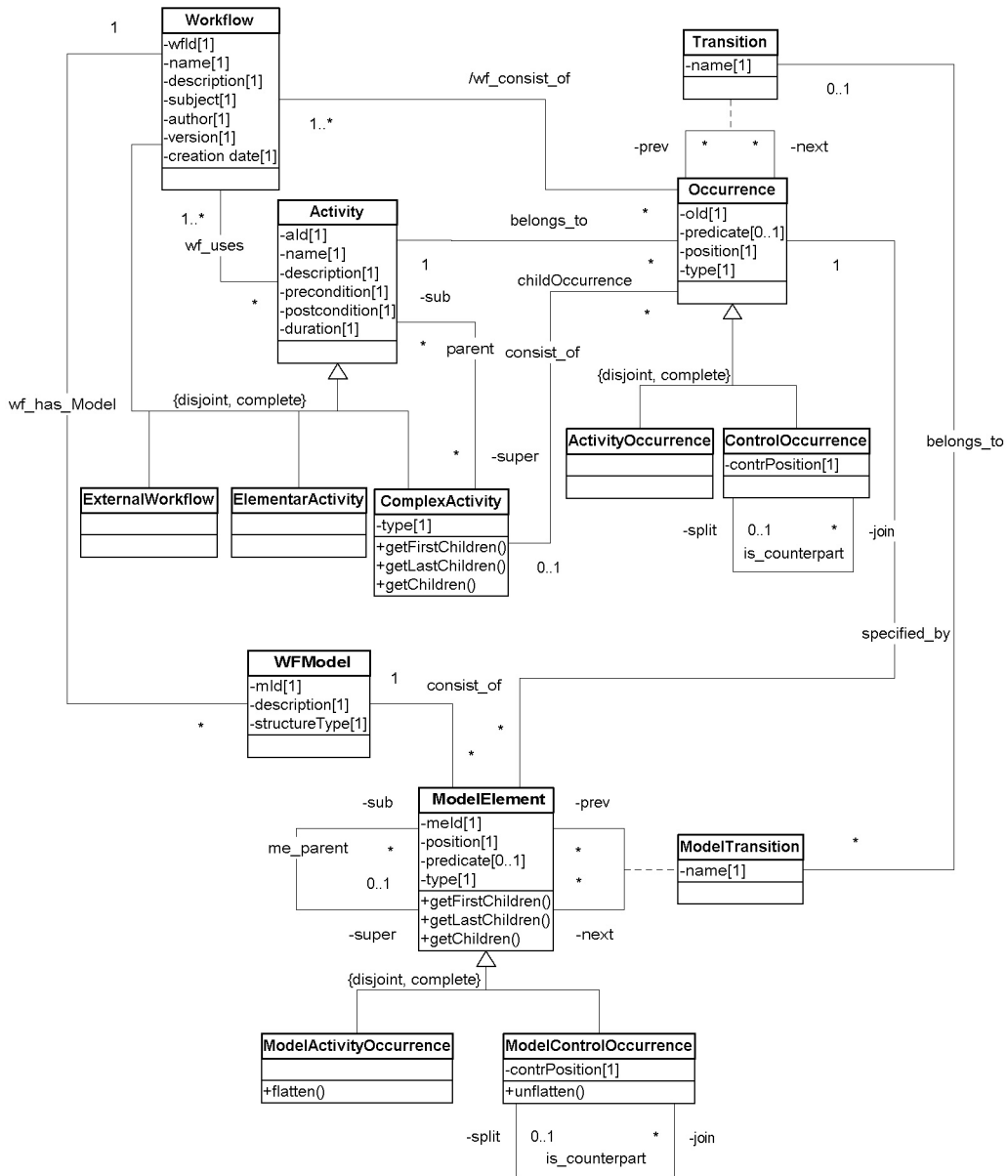


Figure 4.1: Workflow Metamodel

either expressed by the generalization hierarchy. An activity may belong to many workflows, a fact that is modeled by the association *wf_uses*.

4.4.2.2 Complex Activity

Complex activities, modeled by the class *ComplexActivity*, are composed of other activities, which are represented as (activity-) occurrence in the composition of a complex activity. Complex activities have a type (attribute *type*) describing their control structure (*seq* for a sequential activity, *par* or *and* for a parallel activity, *cond* or *or* for a conditional activity or *alt* for an alternative activity). Through the association *parent* between activities and complex activities (hierarchical relationship) we also register in which activities an activity appears.

4.4.3 Occurrences

To support reuse the notion of occurrence is central in our metamodel. Occurrences are specified by the class *Occurrence* which has the attributes *oID* (unique), *predicate* (optional), *position* and *type*. The attribute *predicate* is used to represent the condition for child occurrences of conditional activities and for occurrences following an or-split. The attribute *position* indicates the processing position within the scope of the complex activity, and it can have either of the values ‘start’, ‘between’, ‘split’, ‘join’, ‘end’ or ‘start/end’. The attribute *type* shows the same information as the attribute *type* of the class *ComplexActivity*. An occurrence can either be an activity occurrence or a control occurrence, as described below. Each occurrence corresponds to exactly one activity by the association *belongs_to*, and each activity corresponds to several occurrences.

An occurrence is the placeholder for an activity, i.e., for each complex activity the occurrences of its child activities are determined by the association *consist_of*. These occurrences are designated as child occurrences of the complex activity.

For all child occurrences of a sequence activity, the predecessor and successor for each of this occurrence is defined by means of transitions which are modeled by the association class *Transition* where the association is a reflexive relationship referencing the class *Occurrence*. Per sequence, it produces exactly one child occurrence *S* that has no predecessor, and exactly one child occurrence *J* that has no successor. If *S* and *J* are the same occurrence, the attribute *position* of *S* holds the value ‘start/end’. Otherwise *S* has the value ‘start’ and *J* the value ‘end’. The remaining child occurrences (if existent) have the value ‘between’. It has to be noted that transitions can be defined only for child occurrences of sequences and control structures, since predecessors and successors can only be defined for occurrences.

Every child occurrence of conditional activities and every occurrence that follows an or-split is associated with a predicate, so that there is exactly one occurrence with which the predicate is fulfilled in the processing.

All child occurrences of conditional, parallel and alternative activities have the value ‘start/end’ in the attribute *position*.

At this point the description of the class *ComplexActivity* can be finished. Apart from the attributes mentioned above, *ComplexActivity* has the methods *getFirstChildren*, *getLastChildren* and *getChildren*, where *getFirstChildren* returns the child occurrences of the complex activity with the value ‘start’ or ‘start/end’ of the attribute *position*, *getLastChildren* returns the child occurrences of the complex activity with the value ‘end’ or ‘start/end’ and *getChildren* returns all child occurrences of the complex activity. There are some other reasonable methods concerning this class (cf. [Eder et al., 2003; Ninaus, 2002]).

4.4.3.1 Activity Occurrence

The occurrence of an activity in the specification appears both in the class *ActivityOccurrence* and *Occurrence*. The class *ActivityOccurrence* inherits the characteristics from the class *Occurrence*. The class *ActivityOccurrence* has the method *flatten* that carries out the process described below.

4.4.3.2 Control Occurrence

A control occurrence represents a control element such as a split or a join (see above). The class *ControlOccurrence* that has the attribute *cntrPosition*, models these control elements and inherits the characteristics from the class *Occurrence*. Moreover, it has the method *unflatten*. The attribute *cntrPosition* indicates whether it belongs to a split or a join control element. The class *ControlOccurrence* has the reflexive (recursive) association *is_counterpart* that indicates the accompanying (matching) join control element to the split control element and vice versa.

4.4.4 Workflow Model

The description of workflow models is contained in the class *WFModel* that has the attributes *mId* (unique), *description* and *structureType*. The attribute *structureType* indicates whether the workflow is a strict blocked workflow or a hybrid workflow. This class is connected by the association *wf_has_Model* to the class *Workflow*. This relationship is needed, since a workflow can have several models.

4.4.4.1 Model Element

A workflow model consists of model elements that are represented by the class *ModelElement* with the attributes *meId* (unique), *predicate* and *position*. The class

ModelElement has the methods *getFirstChildren*, *getLastChildren* and *getChildren* (they are analogous to the methods of *ComplexActivity* with identical names). An object of the class *ModelElement* references exactly one object of the class *Occurrence* by the association *specified_by*. An object of the class *Occurrence* references many objects of the class *ModelElement*. The attribute *position* is analogous to the attribute of the class *Occurrence* with identical name. An element can either be an activity occurrence or a control occurrence. Analogous to occurrences, model elements can have transitions, modeled by the class *ModelTransition*. The hierarchical relationship between model elements is maintained by the association *me_parent*.

4.4.4.2 Model Activity Occurrence

The occurrence of an activity in the model places a model occurrence there, and it is modeled by the class *ModelActivityOccurrence*. The class has the method *flatten* that carries out the process described in the subsequent Chapter.

4.4.4.3 Model Control Occurrence

If a model occurrence that represents a complex activity is flattened, the model occurrence is represented through a split and a join control element, the so-called control occurrences. These control elements are modeled by the class *ModelControlOccurrence* that has the attribute *cntrPosition* and the method *unflatten*. The attribute *cntrPosition* indicates whether it concerns to a split or a join control element. The method *unflatten* is the inverse method to *flatten* of the class *ModelActivityOccurrence*. The class *ModelControlOccurrence* has the recursive association *is_counterpart* that indicates the accompanying join control element to the split control element and vice versa.

4.4.5 Metamodel Integrity Constraints

Integrity constraints provide a way of ensuring that changes made to the workflow model by authorized users do not result in a loss of workflow consistency. The violation of consistency constraints results in erroneous system behavior and has therefore to be avoided. Thus, integrity constraints form an essential part of the workflow metamodel. A declaratively specified integrity constraint can be any arbitrary predicate applied to the workflow model. It is customary to distinguish between two kinds of constraints: static and dynamic [Chomicki, 1995]. Only static constraints are considered here. They refer to the current state of the workflow model (e.g., “all child occurrences of a complex activity of the type ‘seq’ must be interrelated”).

In the following, the integrity constraints are described in two ways, i.e. formless in prose and formally in OCL. Table 4.1 gives a short overview of the integrity constraints that are not represented in the class diagram.

Table of Integrity Constraints		
Level	Name	Description
Specification	IC1	Every complex activity must have at least one child-activity
	IC2	Every complex activity must have at least one child-occurrence
	IC3	All child-activities of an activity must be referenced at least by a child-occurrence of the activity
	IC4	All child-occurrences of an activity referring to exactly one child-activity of the activity
	IC5	Sequence-activities must have successional child-occurrences
	IC6	Only child-occurrences of sequence-activities are allowed to have transitions
	IC7	Corresponding split and join control occurrences (which belong together) reference mutually (to each other)
	IC8	The corresponding split and join control occurrences reference the same occurrence
	IC9	The relation given by the association parent is acyclic
Model	IC10	If a model exists, every occurrence has at least one dedicated model-occurrence
	IC11	If a model exists, the model must be complete and sound
	IC12	If a model exists, the model occurrence transitions are analogous to the occurrence transitions
	IC13	If a model exists, the hierarchical structure of the model occurrences has to be analogous to the hierarchical structure of the occurrences
	IC14	Corresponding split and join model control occurrences (which belong together) reference mutually (to each other)
	IC15	The corresponding split and join model control occurrences reference the same occurrence
	IC16	The transitions of model control occurrences must be analogous to the structure of the corresponding complex activity
	IC17	The relation given by the association me_parent must be acyclic

Table 4.1: Integrity Constraints (Brief Description)

4.4.5.1 Integrity Constraints Belonging the Specification Level

IC1: A complex activity is composed of sub-activities and must therefore have at least one child-activity. Thus, the set of sub-activities of a complex activity must be not empty.

ComplexActivity:

```
self.sub->notEmpty
```

IC2: A complex activity consists of occurrences and must therefore have at least one child-occurrence. Thus, the set of child-occurrences of a complex activity must be not empty.

ComplexActivity:

```
self.childOccurrence->notEmpty
```

IC3: All child-activities of an activity must be referenced by a child-occurrence of the activity and vice versa. Hence, the set of sub-activities of a complex activity *CA* has to be identical with the set of activities which are referenced by the child-occurrences of *CA* and vice versa. This requires that IC1 and IC2 are valid.

ComplexActivity:

```
self.sub->includesAll(self.childOccurrence.activity) and  
self.childOccurrence.activity->includesAll(self.sub)
```

IC4: Every occurrence belongs to exactly one activity. Thus, the cardinality of the set of activities of an occurrence is one.

Occurrence:

```
self.activity->size = 1
```

IC5: The child-occurrences of a complex activity of type ‘seq’ must have successional (interrelated) child-occurrences, such that every child-occurrence has one of these child-occurrences as successor, except the child-occurrence with the position ‘end’ or ‘start/end’, and such that every child-occurrence has one of these child-occurrences as predecessor, except for the child-occurrence with the position ‘start’ or ‘start/end’. In order to apply this integrity constraint, it is necessary that IC1, IC2, IC3 and IC4 are valid.

ComplexActivity:

```
-- There is exactly one 'start'-occurrence

self->select(type ='seq').childOccurrence->select(
  position ='start' or position ='start/end')->size = 1

-- There is exactly one 'end'-occurrence
self->select(type ='seq').childOccurrence->select(
  position ='end' or position ='start/end')->size = 1

-- Every occurrence except the end-occurrence has
-- exactly one successor

self->select(type ='seq').childOccurrence->select(
  position ='start' or position ='between')->forAll(
  hs | hs.next->size = 1)

-- The end-occurrence has no successor

self->select(type ='seq').childOccurrence->select(
  position ='end' or position ='start/end').next->isEmpty

-- The corresponding activity of the successor-occurrence
-- of an occurrence and the corresponding activity of
-- the occurrence are identical

self->select(type ='seq').childOccurrence->select(
  position ='start' or position ='between')->forAll(
  hs | hs.activity = hs.next.activity)

-- Every occurrence except the start-occurrence has
-- exactly one predecessor

self->select(type ='seq').childOccurrence->select(
  position ='between' or position ='end')->forAll(
  hp | hp.prev->size = 1 }

-- The start-occurrence has no predecessor

self->select(type ='seq').childOccurrence->select(
  position ='start' or position ='start/end').prev->isEmpty

-- The corresponding activity of the predecessor occurrence
-- of an occurrence and the corresponding activity of
-- the occurrence are identical

self->select(type ='seq').childOccurrence->select(
  position ='between' or position ='end')->forAll(
  hp | hp.activity = hp.activity.prev }
```

IC6: Only child-occurrences of sequence-activities are allowed to have transitions, i.e. the predecessor and successor sets of child-occurrences of complex activities with the type 'par', 'cond' or 'alt' are empty.

ComplexActivity:

```
self->select(type = 'par').childOccurrence.next->isEmpty and
self->select(type = 'par').childOccurrence.prev->isEmpty

self->select(type = 'cond').childOccurrence.next->isEmpty and
self->select(type = 'cond').childOccurrence.prev->isEmpty

self->select(type = 'alt').childOccurrence.next->isEmpty and
self->select(type = 'alt').childOccurrence.prev->isEmpty
```

IC7: Corresponding (matching) split and join control occurrences reference mutually to each other. A split control occurrence (cntrPosition = 'start') has at least one matching join control occurrence (cntrPosition = 'end') and a join control occurrence has exactly one matching split control occurrence. Other relationships between control occurrence are not allowed.

ControlOccurrence:

```
self.cntrPosition = 'start' implies
self->includesAll(self.join.split) and self.split->isEmpty

self.cntrPosition = 'end' implies self = self.split.join and
self.join->isEmpty
```

IC8: Corresponding (matching) split and join control occurrences reference the same super-activity. This means that they have to stem from one complex activity. This, the super-activity of the split control occurrence is the same activity as the super-activity of the join control occurrence and vice versa. This constraint requires that IC7 is valid.

ControlOccurrence:

```
self.cntrPosition = 'start' implies
self.activity.super = self.join.activity.super

self.cntrPosition = 'end' implies
self.activity.super = self.split.activity.super
```

IC9: The relation given by the association parent must be acyclic. Hence a complex activity does not contain itself as an (transitive) sub-activity.

Activity:

```
not self.getAllSubActivities()->includes(self)
```

4.4.5.2 Integrity Constraints Belonging to the Model Level

IC10: For every model, every occurrence of the specification from which the model is derived, has at least one dedicated model occurrence in the model. That means that if a model *WFM* is derived from a specification *WFS*, every occurrence of *WFS* has at least one pendant in *WFM*. This constraint assumes that the specification is correct.

Workflow:

```
self.wfModel->forAll( wfm | wfm.modelElement->notEmpty implies
  self.occurrence->forAll( so |
    wfm.modelElement.occurrence->exists(elem | elem = so)))
```

IC11: If a model exists, then the model must be complete and sound, i.e. the model must be correctly expanded according the specification, such that an occurrence of an activity has to be aware of its process context. Thus, if a model *WFM* is derived from a specification *WFS*, every model element *ME*, which belongs to a complex activity *CA*, features the analogous sub-model elements as *CA* (child-occurrences of *CA*). This constraint requires that IC9 is valid.

Workflow:

```
self.wfModel->forAll( wfm | wfm.modelElement->notEmpty implies
  wfm.modelElement->forAll(me |
    me.occurrence.activity.childOccurrence->forAll(so |
      so.modelElement->exists(me1 | me1.super = me)))
```

IC12: If a model exists, then the model occurrence transitions are analogous to the occurrence transitions of the specification. Hence, if is a model element *ME*, which is referenced by a specification occurrence *SO*, has a predecessor model element *PME*, then the referenced specification occurrence *PSO* of *PME* must be the predecessor of *ME*. Likewise, if a model element *ME*, which is referenced by a specification occurrence *SO*, has a successor model element *SME*, then the referenced specification occurrence *SSO* of *SME* must be the successor of *ME*. This constraint requires that IC9 and IC10 are valid.

Occurrence:

```
self.prev.modelElement->forAll(mo | self.modelElement.prev = mo) and
self.next.modelElement->forAll(mo | self.modelElement.next = mo)
```

IC13: If a model exists, then the hierarchical structure of the model elements has to be analogous to the hierarchical structure of the occurrences of the underlying specification so, for all model elements *ME*, which reference a complex activity *CA*, the referenced set of specification occurrences of their sub-model elements is identical to the set of the child-occurrences of *CA*.

ModelElement:

```
self->select(
  me | me.occurrence.activity.ocIsKindOf(ComplexActivity) = true)->forAll(
  me1 | me1.sub.occurrence->includesAll(
    me1.occurrence.activity.childOccurrence))
```

IC14: Corresponding split and join model control occurrences (which belong together) reference mutually (to each other). A split model control occurrence (cntrPosition = 'start') has at least one matching join model control occurrence (cntrPosition = 'end') and a join model control occurrence has exactly one matching split model control occurrence. No other relationships between model control occurrences are allowed.

ModelControlOccurrence:

```
self.cntrPosition = 'start' implies
self->includesAll(self.join.split) and
self.split->isEmpty

self.cntrPosition = 'end' implies self = self.split.join and
self.join->isEmpty
```

IC15: Corresponding split and join model control occurrences reference the same occurrence. This means that they have to stem from one complex activity. Therefore, the occurrence which is referenced by a split model control occurrence, is the same occurrence as the occurrence which is referenced by the matching join model control occurrence and vice versa.

```
ModelControlOccurrence:
```

```
self.cntrPosition = 'start' implies
self.occurrence = self.split.occurrence
```

```
self.cntrPosition = 'end' implies
self.occurrence = self.join.occurrence
```

IC16: The transitions of model control occurrences must be analogous to the structure of the corresponding complex activity. Hence a split has a transition to every model element, which returns the method `getFirstChildren()` applied to the split, and a join has a transition to every model element, which returns the method `getLastChildren()` applied to the join.

```
ModelControlOccurrence:
```

```
self.cntrPosition = 'start' implies
self.next->includesAll(self.getFirstChildren())
```

```
self.cntrPosition = 'end' implies
self.prev->includesAll(self.getLastChildren())
```

IC17: The relation given by the association `me_parent` must be acyclic. This means that a model element does not contain itself as a (transitive) sub-model element.

```
ModelElement:
```

```
not self.getAllSubElements()->includes(self)
```

4.5 Workflow Example

In the course of the thesis we will now use the following running example in Figure 4.2.

In this graph, we have a sequence of activity occurrence *A*, a parallel control structure with and-split *M2* and *M9* as the corresponding and-join, and the activity occurrence *I*. The parallel structure has two paths with the activity occurrence *H* and a sequence (*M3* and *M8*). This sequence consists of activity occurrence *B*, a conditional structure and activity occurrence *G*. The conditional structure has three paths with activity occurrence *E*, *F* and a sequence, which is composed of activity occurrence *C* and *D*.

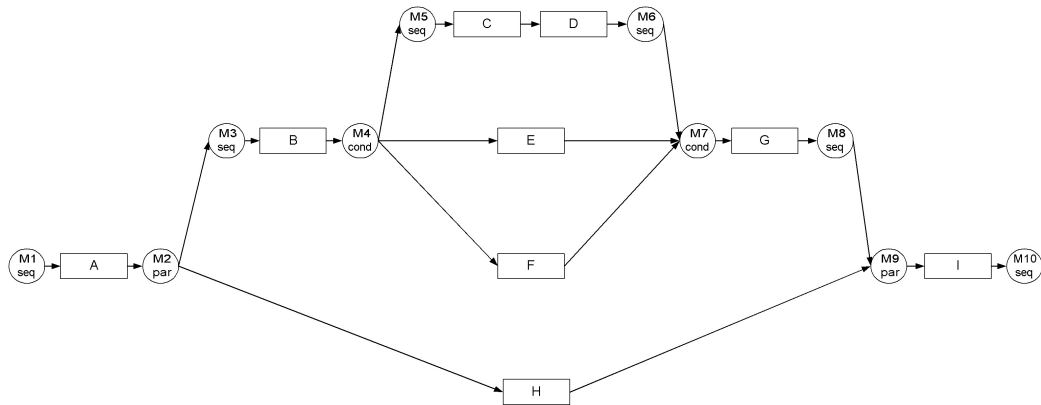


Figure 4.2: Example Workflow

4.6 Summary

This Chapter starts out with a couple of requirements for a workflow metamodel, which depend on the process perspective, are pointed out. Then, the metamodel for workflow definitions that supports control structure oriented as well as graph based representation of processes is presented. It is pointed out that the elaborated hierarchical composition supporting re-use of activity definitions and the separation of specification and model level in the workflow description are important aspects of this metamodel. Moreover, metamodel constraints that have to be fulfilled in order to guarantee the integrity of our workflow model are presented.

5

Workflow Transformations

In this Chapter, workflow transformations which do not change the semantics of the workflow are considered in particular. Such transformation operations may be applied to a process model SWF to transform it into SWF', so that SWF and SWF' still maintain an underlying structural relationship with each other.

This Chapter is organized as follows: Section 5.1 introduces the notion of workflow transformations. Section 5.2 gives a brief overview of related work in literature. Sections 5.3 and 5.4 capture necessary definitions for the subsequent Sections. Sections 5.5 and 5.6 show the transformations in detail. Section 5.7 and 5.8 conclude this Chapter.

5.1 Introduction

Generally, a workflow model evolves through numerous changes during its lifetime to meet dynamic and changing business requirements. New technology, new laws, and new market requirements lead to modifications of the workflow process definitions at hand. Organizations are continually faced with the challenge to bring ideas and concepts to products and services in an ever-increasing pace. Companies separated by space, time and capabilities come together to deliver products and solutions required by the global marketplace. The trends towards virtual corporations and e-commerce, and increasing global networking of economies have become a reality. As a result, more and more workflow processes are subject to continuous change¹ [van der Aalst, 2001; Ellis and Keddara, 2000; van der Aalst, 1999; Joeris and Herzog, 1998; Casati et al., 1996b 1995].

A critical challenge for workflow management systems is their ability to respond effectively to changes [Ellis and Keddara, 2000; Reichert and Dadam, 1998].

¹Adaptive workflow is an area of research which examines concepts, techniques, and tools to support change.

Changes may range from ad-hoc modifications of the process for a single customer to a complete restructuring of the workflow process to improve efficiency.

5.1.1 Types of Workflow Changes

This Section deals with the different kinds of changes and their consequences. Some of the perspectives relevant for change are [van der Aalst et al., 1999]:

- **process perspective**, i.e. tasks are added or deleted or their ordering is changed.
- **resource perspective**, i.e. resources are classified in a different way, or new classes are introduced.
- **control perspective**, i.e. changing the way resources are allocated to processes and tasks.
- **task perspective**, i.e. upgrading or downgrading tasks.
- **system perspective**, i.e. changes to the infrastructure or the configuration of the engines in the enactment service.

5.1.2 Process Perspective Changes

Two kinds of changes in the process perspective are identified [van der Aalst et al., 1999]:

- **individual (ad-hoc) changes** are handled on a case-by-case basis. In order to provide customer specific solutions or to handle rare events or errors, the process is adapted for a single case or a limited group of cases. Exceptions often result in ad-hoc changes. A typical example of an ad-hoc change is skipping a task in case of emergency. This kind of change is often initiated by some external factor. A typical dilemma related to ad-hoc change is the problem to decide what kinds of changes are allowed and the fact that it is impossible to foresee all possible changes.
- **structural (evolutionary) changes** of the workflow process: from a certain moment in time, the process changes for all new cases to arrive at the system. A structural change is normally the result of a new business strategy, re-engineering efforts (BPR), or a permanent alteration of external conditions (e.g., a change of law). An evolutionary change is initiated by the management to improve efficiency or responsiveness, or it is forced by legislature or changing market demands. The workflow is changed to improve responsiveness to the customer or efficiency (do more with less).

Both for ad-hoc and evolutionary changes, we distinguish three ways in which the routing of cases along tasks can be changed [van der Aalst et al., 1999]:

- **Extend:** adding new tasks which (1) are executed in parallel, (2) offer new alternatives, or (3) are executed in-between existing tasks.
- **Replace:** a task is replaced by another task or a subprocess (i.e., refinement), or a complete region is replaced by another region.
- **Re-order:** changing the order in which tasks are executed without adding new tasks, e.g. swapping tasks or making a process more or less parallel.

Generally, workflow transformations are operations on a workflow SWF resulting in a different workflow SWF' . Each workflow transformation deals with a certain aspect of the workflow (e.g. move splits or joins, eliminate a hierarchy level) [Eder and Gruber, 2002].

It is essential that such changes are introduced systematically and that their impact is clearly understood. Workflow model transformation is a suitable approach for this purpose [Sadiq and Orłowska, 2000]. The application of pre-defined transformation operations can ensure that the modified process conforms to a given class of constraints specified in the original model.

5.1.3 Classes of Transformation

Obviously, two classes of transformation principles can be identified to capture evolving changes during the lifetime of workflows. The first one is the class of *equivalence-preserving transformations*. Here, much attention must be paid to the equivalence criterion, since there are different notions on the equivalence criterion which biases the set of transformations. The second one is the class of *non-equivalent transformations*. Several sub-classes of this class of transformation can be subsumed to it with different degrees of restrictiveness. The most unrestricted class is the class of arbitrary transformations where no restrictions to it exist. In [Sadiq and Orłowska, 2000] for instance, two classes of non-equivalent transformations are introduced, namely the imply and subsume class with progressively relaxed restrictions.

We focus on structurally equivalent transformations. In the following we develop a set of basic transformations which do not change the semantics of the workflow. For this purpose, we introduced a new equivalence criterion on workflows and workflow instance types. Furthermore we established complex transformations based on this basic set of transformations by repeated application of those. These complex transformations allow to separate the intrinsic instance types of a

workflow. All presented transformations are feasible in both directions (symmetric), i.e. from SWF to SWF' and vice versa from SWF' to SWF [Eder and Gruber, 2002].

5.2 Related Work

So far, there has not been much literature about workflow transformations. In [van der Aalst et al., 2000] various workflow patterns for different WfMSs concerning different workflow models are catalogued. The alternative representations employ different control elements. The alternative representations are assumed to be semantically equivalent, but there is no equivalence criterion, nor there are any transformation rules.

[Kiepuszewski et al., 1999] address the modeling of structured workflows and transforming arbitrary models to structured models, based on the equivalence notion of *bisimulation*. In this paper, the authors investigate transformations based on several patterns, and analyze in which situations transformations can be applied. One of the specified transformations is moving split nodes, which is, in contrast to our work, considered a non-equivalent transformation. The so-called *overlapping structure*, which has been introduced in the context of workflow reduction for verification purposes, is adopted in our work and it is used by the transformation *Moving and-join over or-join*.

Finally, in [Sadiq and Orłowska, 2000], three classes of transformation principles are identified to capture evolving changes of workflows during their lifetime. We only focus on the first class, that is on structurally equivalent transformations. In this work, the equivalence criterion (relationship) for structurally equivalent workflows is too restrictive, because the workflows must have identical sets of *execution nodes*, which implies that transformations using node duplication cannot be applied. Considering the differences in the workflow models we adopted the eliminating of join nodes as join coalescing with different semantics.

5.3 Workflow Instance Type

Due to conditionals not all instances of a workflow processes the same activities. We classify workflow instances into workflow instance types according to the activities actually executed. Similar to [Marjanovic and Orłowska, 1999b], a *workflow instance type* refers to (a set of) workflow instances that contain exactly the same activities, i.e. for each or-split node in the workflow graph the same successor node is chosen; for each conditional complex activity the same child-activity is selected. Therefore, a workflow instance type is a submodel of a workflow where each or-split has exactly one successor; each conditional or alternative complex activity has exactly one subactivity [Eder and Gruber, 2002].

Fig. 5.1 shows the instance types of the above workflow in Fig. 3.11 in textual notation. The graphical representation is depicted in Fig. 5.2.

IT1:InstanceT sequence M1:01 end end	M1:ModelElem sequence M2:02 M3:03 M4:04 end end	M2:ModelElem conditional M5:05 end end	M4:ModelElem conditional M7:05 end end	M3,M5,M7: ModelElem elementary end
IT2:InstanceT sequence M1:01 end end	M1:ModelElem sequence M2:02 M3:03 M4:04 end end	M2:ModelElem conditional M5:05 end end	M4:ModelElem conditional M8:06 end end	M3,M5,M8: ModelElem elementary end
IT3:InstanceT sequence M1:01 end end	M1:ModelElem sequence M2:02 M3:03 M4:04 end end	M2:ModelElem conditional M6:06 end end	M4:ModelElem conditional M7:05 end end	M3,M6,M7: ModelElem elementary end
IT4:InstanceT sequence M1:01 end end	M1:ModelElem sequence M2:02 M3:03 M4:04 end end	M2:ModelElem conditional M6:06 end end	M4:ModelElem conditional M8:06 end end	M3,M6,M8: ModelElem elementary end

Figure 5.1: Workflow instance types in text-based notation

5.4 Equivalence of Workflows

In the previous Sections we presented some of the possibilities to change workflow specifications. Actually these changes only make sense when they can be performed in a way that we can guarantee (beforehand) that the transformation satisfies a certain set of correctness preservation properties. If we were not able to make any statements about correctness preservation, the new system would have no relationship to the old one [van der Aalst et al., 1999].

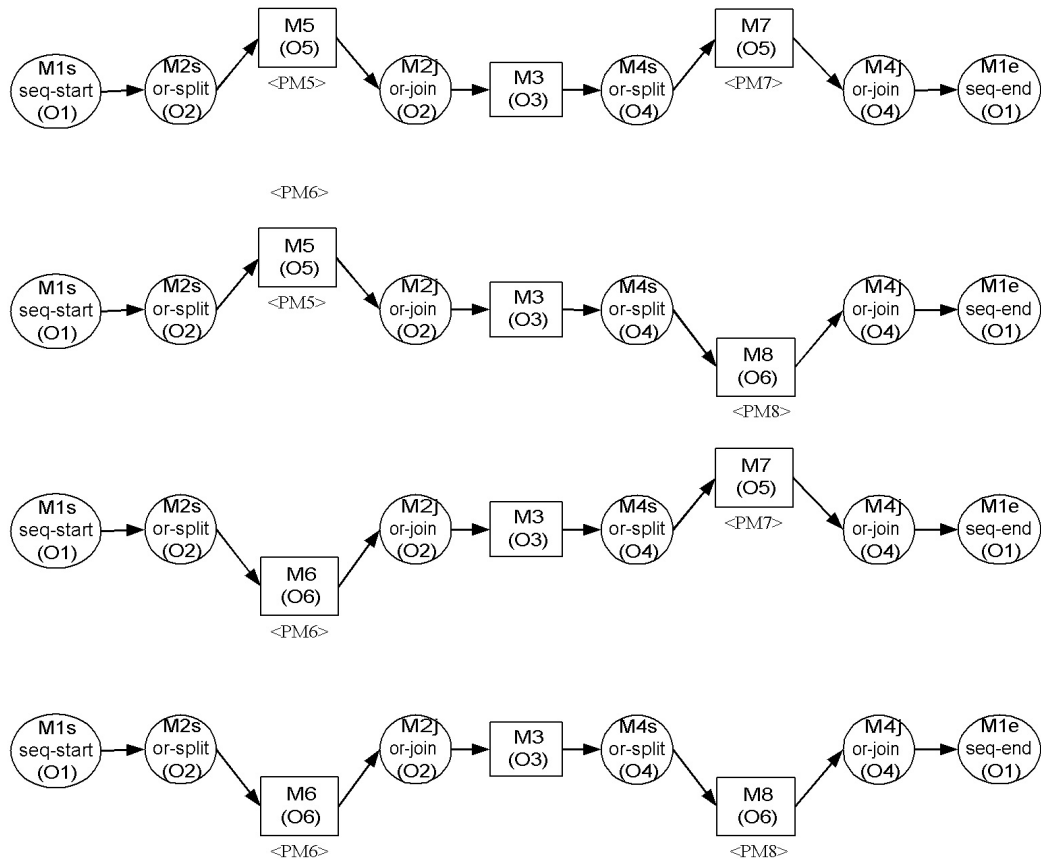


Figure 5.2: Workflow instance types in graph-based notation

Two different kinds of correctness notions can be distinguished, viz. syntactic and semantic correctness.

Syntactic correctness means that the workflow which results from applying a transformation operation is still syntactically correct. This is accomplished by the integrity constraints of the metamodel in the previous Chapter. Syntactic correctness is independent of the context.

Semantic correctness deals with similarities between the capabilities of the old workflow and the capabilities of the new workflow. It may therefore be desirable that the new workflow is semantically equal to the old workflow. Semantic correctness is concerned with the context in which the change occurs.

5.4.1 Equivalence Definitions

Our goal is to support the transformation of workflows without changing the semantics. For this purpose we need a clear definition when workflows are equivalent. Our definition is based on the consideration that workflows are equivalent if they provide the same tasks. Therefore, the equivalence of correct workflows bases on equivalent tasks and its identical execution order. Workflows are equivalent if they execute the same tasks in exactly the same order. Therefore, the equivalence of correct workflows ($WF1 \equiv WF2$) is based on equivalent sets of workflow instance types [Eder and Gruber, 2002].

DEFINITION: Two workflows $WF1$ and $WF2$ are equivalent ($WF1 \equiv WF2$) if their sets of instance types $ITS1$ and $ITS2$ are equivalent ($ITS1 \equiv ITS2$). Two instance type sets are equivalent if and only if for each element of one set $IT1$ there is an equivalent element in the other set $IT2$ ($IT1 \equiv IT2$). (5.1)

DEFINITION: Two workflow instance types $IT1$ and $IT2$ are equivalent ($IT1 \equiv IT2$) if they consist of occurrences of the same (elementary) activities with the identical execution order. The position of or-splits and or-joins in instance types is irrelevant, since an or-split has only one successor in an instance type. (5.2)

DEFINITION: Two occurrences O_1 and O_2 are equivalent ($O_1 \equiv O_2$) if they belong to the same activity A ($O_1.activity = O_2.activity$). (5.3)

5.4.1.1 Equivalence Relation

We can define the equivalence as a relation $\equiv \subseteq WF \times WF$.

Such a relation \equiv is reflexive iff $wf_1 \equiv wf_1$.

Such a relation \equiv is symmetric iff $wf_1 \equiv wf_2$ implies $wf_2 \equiv wf_1$ and $wf_2 \equiv wf_1$ implies $wf_1 \equiv wf_2$.

Such a relation \equiv is transitive iff $wf_1 \equiv wf_2 \wedge wf_2 \equiv wf_3$ implies $wf_1 \equiv wf_3$.

5.4.1.2 Equivalence Example

Figure 5.3 shows a workflow *SWF1* with a sequence *CS* consisting of *M1*, *CC* and *M4*. The conditional element *CC* consists of *M2* and *M3*, both are elementary occurrences. The elementary occurrence *M1* refers to activity *A*, *M2* refers to activity *B*, *M3* refers to activity *C*, and *M4* refers to activity *D*. We can build the instance types of *SWF1* which are illustrated below. *SWF1(IT1)* and *SWF1(IT2)* depicts the two instance types.

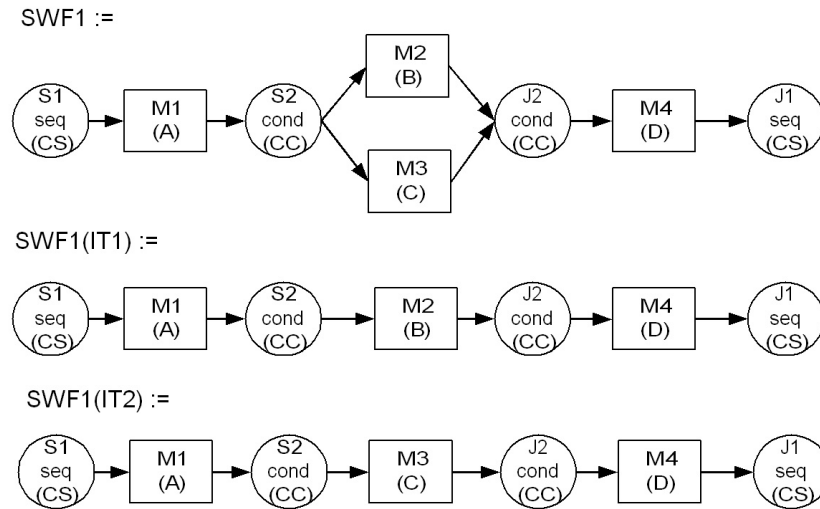


Figure 5.3: Workflow Example with Instance Types - 1

Figure 5.4 shows a similar workflow *SWF2* with a sequence *CS* consisting of *M1* and *CC*. The conditional element *CC* consists of *CS1* and *CS2*, both are sequence structures with occurrences *M2* and *M41* or *M3* and *M42*. The elementary occurrence *M1* refers to activity *A*, *M2* refers to activity *B*, *M3* refers to activity *C*, and *M4* refers to activity *D*. We can build the instance types of *SWF2* which are illustrated below. *SWF2(IT1)* and *SWF2(IT2)* depicts the two instance types.

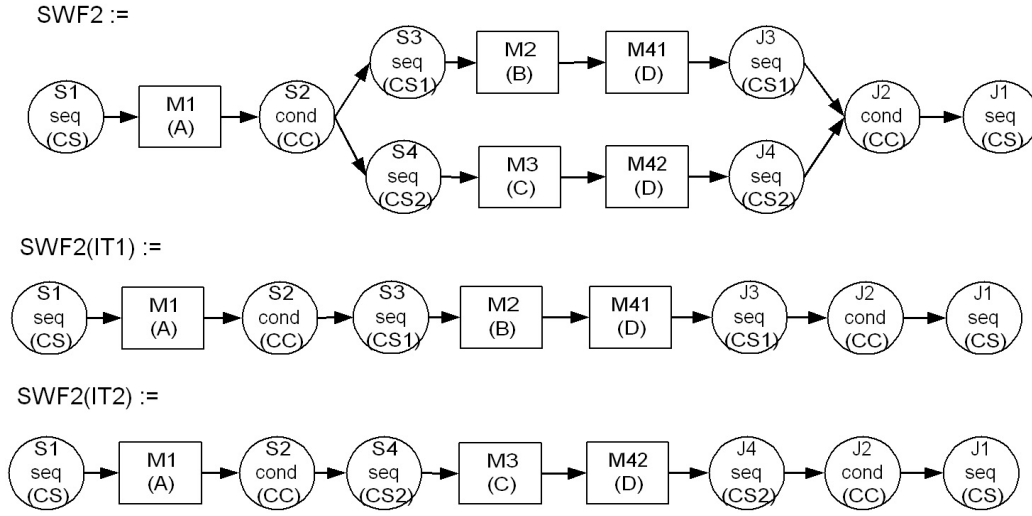


Figure 5.4: Workflow Example with Instance Types - 2

In order to check the equivalence of *SWF1* and *SWF2* we have to check the sets of instance types *ITS1* and *ITS2*. *ITS1* consists of *SWF1(IT1)* and *SWF1(IT2)*, and *ITS2* consists of *SWF2(IT1)* and *SWF2(IT2)*. We take *SWF1(IT1)* which executes the activities *A*, *B* and *D* in this order. Now, there must be an instance type in *SWF2(IT2)* that executes the same activities in identical order. *SWF2(IT1)* executes the activities *A*, *B* and *D* in this order. Hence, both instance types are equivalent. We take the next instance type of *ITS1*, which is *SWF1(IT2)*. It executes the activities *A* (*M1*), *C* (*M3*) and *D* (*M4*) in this order. We take the remaining instance type of *ITS2*, which is *SWF2(IT2)*. It executes the activities *A* (*M1*), *C* (*M3*) and *D* (*M4*) in this order. Therefore, *SWF1(IT2)* and *SWF2(IT2)* are equivalent. Since any instance type of *ITS1* has an equivalent instance type in *ITS2* and vice versa, both workflows are equivalent.

5.4.1.3 Equivalence Transformation Relation

We can define a workflow transformation as a relation $\sim_t \subseteq WF \times WF$.

Such a relation \sim_t is reflexive iff $wf_1 \sim_t wf_1 \Rightarrow wf_1 \equiv wf_1$.

Such a relation \sim_t is symmetric iff $wf_1 \sim_t wf_2 \Rightarrow wf_1 \equiv wf_2$.

Such a relation \sim_t is transitive iff $wf_1 \sim_t wf_2 \wedge wf_2 \sim_t wf_3$ implies $wf_1 \equiv wf_3$.

5.5 Basic Transformations

A basic transformation defines a one-step transformation of a workflow. A basic transformation applies to a workflow if a part of the workflow matches the left-hand or right-hand side pattern of the transformation. In this case the workflow is replaced with the workflow in which the matching part is replaced by the corresponding transformation pattern (right-hand or left-hand side) that were matched in the left-hand or right-hand side. Thus a transformation succeeds to apply if the left-hand resp. right-hand side matches, and fails to apply if this is not the case. In general, a transformation may succeed or fail to apply to a workflow.

In the following Sections, the basic workflow transformations are described verbally, graphically and with pseudocode. There are three types of basic transformations: (1) hierarchy manipulation, (2) moving joins, and (3) moving splits. Table 5.1 gives a short overview of the basic workflow transformations.

Table of Basic Workflow Transformations		
Type	Name	Description
Hierarchy Manipulation	WFT-H1	Flatten/Unflatten
	WFT-H2	Sequence Encapsulation
Moving Join	WFT-J1	Join Moving Over Activity Occurrence
	WFT-J2	Join Moving Over Seq-Join
	WFT-J3	Moving Or-Join Over Or-Join
	WFT-J4	Moving Alt-Join Over Alt-Join
	WFT-J5	Moving Or-Join Over Alt-Join
	WFT-J6	Moving Alt-Join Over Or-Join
	WFT-J7	Or-/Alt-Join Coalescing
	WFT-PS	Separating a Conditional/Alternative Path
	WFT-J8	Moving Or-/Alt-Join Over And-Join - Unfold
	WFT-J9	And-Join Moving Over Or-Join
Moving Split	WFT-S1	Moving Or-/Alt-Split Before Activity Occurrence
	WFT-S2	Split Moving Over Seq-Join

Table 5.1: Overview of Basic Workflow Transformations

5.5.1 Hierarchy Manipulation

This type of transformation manipulates the hierarchy relationship of occurrences within a workflow by eliminating (flattening) or adding (unflattening) a level of the

composition hierarchy in a model. Subsequently, the (total) (un)flatten operation is described.

5.5.1.1 Flatten/Unflatten (WFT-H1)

The operation *flatten* eliminates a level of the composition hierarchy in a model by substituting an occurrence of a complex activity by its child occurrences and two control elements (split and join element). Between the split control element and every child occurrence, a dependency is inserted, so that the split element is the predecessor of the child occurrences. Between the last child occurrence(s) and the join control element a dependency is inserted as well, so that the join element is the successor of the child occurrence. Fig. 5.5 shows an example of such a transformation. Here, applying the transformation *flatten* in the workflow model *SWF* on occurrence *M1* with the child occurrences *M2* and *M3*, results in the workflow *SWF'*, where *M1* is replaced by the split *S1* and the join *J1*. *S1* is the predecessor of *M2* and *M3*, and *J1* is the successor of *M2* and *M3*. The application of the *flatten* operation on a workflow model until no further hierarchy can be eliminated is called *total flatten* (see Fig. 3.12). The inverse function to *flatten* is called *unflatten*.

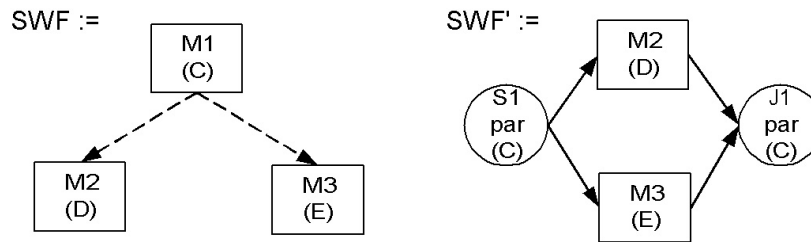


Figure 5.5: Flatten

Algorithm 5.1 describes the flatten operation in pseudocode. Here, the method *insertModelControlOccurrence()* creates a new split control object, *insertModelTransition()* creates a new transition object, and *removeModelElement()* deletes a model element object. The input variable must be an object of the class *ModelActivityOccurrence*.

5.5.1.2 Encapsulation in a Sequence (WFT-H2)

The occurrence of an activity can be encapsulated between two sequence control elements (split and join element). A dependency is inserted between the split control element and the occurrence, so that the split element is the predecessor of the occurrences. Furthermore, a dependency is inserted between the occurrence and

Algorithm 5.1 Flatten

```

1: Procedure Flatten
2: in: oc:ModelActivityOccurrence
3: soc := insertModelControlOccurrence('split')
4: eoc := insertModelControlOccurrence('join')
5: for all element  $e$  in oc.getFirstChildren() do
6:   insertModelTransition(soc, e)
7: end for
8: for all element  $e$  in oc.getLastChildren() do
9:   insertModelTransition(e, eoc)
10: end for
11: for all element  $e$  in oc.sub do
12:   e.super := soc
13: end for
14: insertModelTransition(oc.pred, soc)
15: insertModelTransition(eoc, oc.next)
16: removeModelElement(oc)

```

the join control element, so that the join element is the successor of the occurrence. This new hierarchy level is settled in.

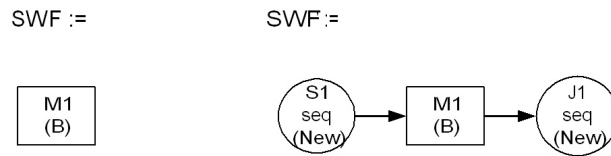


Figure 5.6: Occurrence Encapsulation

Fig. 5.6 shows an example of such a transformation. Here, the application of the transformation in the workflow model SWF on occurrence $M1$ results in the workflow SWF' , where $M1$ is encapsulated between the split $S1$ and the join $J1$. $S1$ is the predecessor of $M1$, and $J1$ is the successor of $M1$.

5.5.2 Moving Joins

Moving Joins means changing the topological position of a join control element (and-join, or-join, alt-join, or seq-join). This transformation separates the intrinsic instance types contained in a workflow model. Some of the following transformations require node duplication. In some cases moving a join element makes it necessary to move the corresponding split element as well.

5.5.2.1 Join Moving Over Activity Occurrence (WFT-J1)

A workflow *SWF* with an or-join or an alt-join *J1* followed by activity occurrence *M3* can be transformed to workflow *SWF'* through node duplication, so that the join *J1* is delayed after *M3* as shown in Fig. 5.7. Here, *M3* will be replaced by its duplicates *M31* and *M32*, so that *J1* is the successor of *M31* and *M32*, and *M1* is the predecessor of *M31* and *M2* is the predecessor of *M32*. This transformation, and all of the following ones, can be applied to structures with any number of paths.

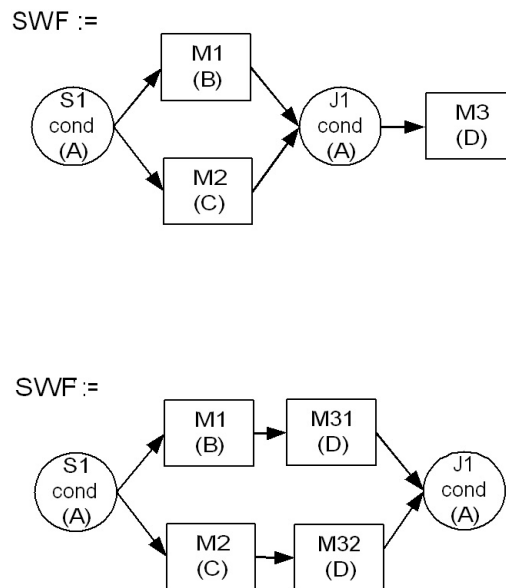


Figure 5.7: Join Moving Over Activity

Algorithm 5.2 describes the transformation operations in pseudocode. Here, the method *copyModelElement(in)* creates a copy of the object *in*. The remaining methods are describe above. The input variable must be an object of class *Model-ControlOccurrence*.

Algorithm 5.2 Join moving over activity

```

1: Procedure Join Moving Over Activity
2: in: j1: ModelControlOccurrence
3: for all po in j1.prev do
4:   nme := copyModelElement(j1.next)
5:   insertModelTransition(nme, j1)
6:   insertModelTransition(po, nme)
7:   removeModelTransition(po, j1)
8: end for
9: te := j1.next
10: j1.next := j1.next.next
11: removeModelElement(te)

```

5.5.2.2 Join Moving Over Seq-Join (WFT-J2)

A workflow SWF with an or-join or an alt-join $J1$ followed by a sequence join $J2$ can be transformed to workflow SWF' through node duplication, so that the join $J1$ is delayed after $J2$ as shown in Fig. 5.8. Here, $J2$ will be replaced by its duplicates $J21$ and $J22$, so that $J1$ is the successor of $J21$ and $J22$, and $M1$ is the predecessor of $J21$ and $M2$ is the predecessor of $J22$. This transformation results in a less structured workflow.

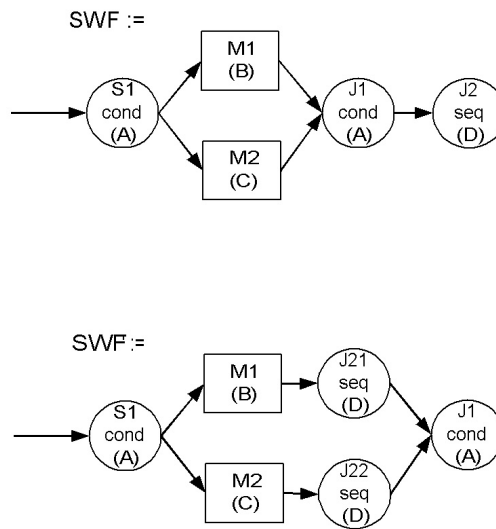


Figure 5.8: Join Moving Over Seq-Join

5.5.2.3 Moving Or-Join Over Or-Join (WFT-J3)

In a workflow SWF with a nested or-structure (i.e. within an or-structure with the split $S1$ and the corresponding join $J1$ there is another or-structure with the split $S2$ and the corresponding join $J2$), the inner join $J2$ can be moved behind the outer join $J1$, which makes it necessary to move the corresponding split element $S2$ and to adjust the predicates according to the changed sequence of $S1$ and $S2$ by conjunction or disjunction. This change causes the inner or-structure to be put over the outer. An example of this transformation in the workflow SWF' is shown in Fig. 5.9.

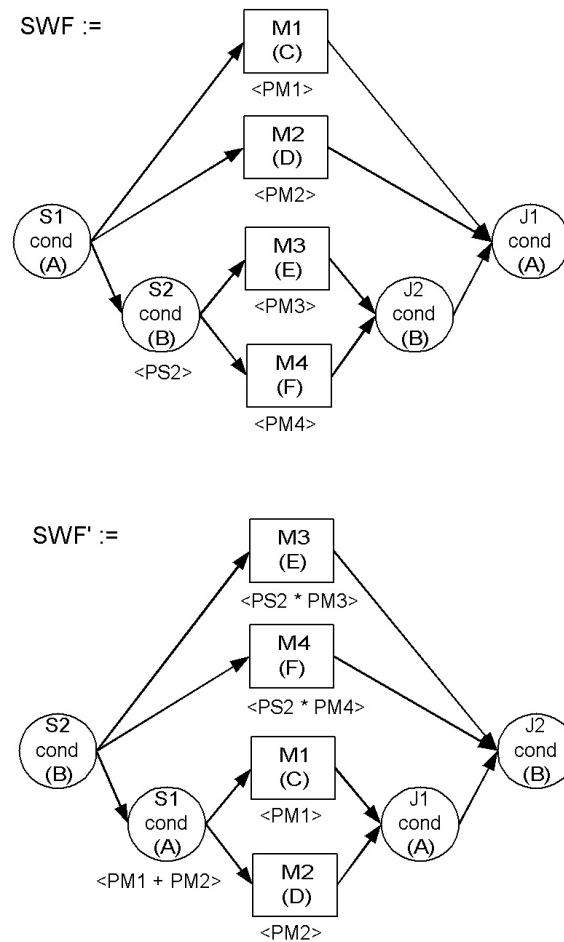


Figure 5.9: Or-Join Moving Over Or-Join

Algorithm 5.3 describes the transformation operations in pseudocode. The input variable must be an object of the class *ModelControlOccurrence*.

Algorithm 5.3 Or-Join Moving Over Or-Join

```
1: Procedure Or-Join Moving Over Or-join
2: in: j1:ModelControlOccurrence
3: s1 := j1.split
4: j2 := j1.prev.oclIsKindOf(ModelControlOccurrence)
5: s2 := j2.split
6: j1.prev := j1.prev - j2
7: j1.next := j2
8: j2.prev := j2.prev  $\cup$  j1
9: j2.next := j1.next
10: s1.next := s1.next - s2
11: s1.prev := s2
12: s2.next := s2.next  $\cup$  s1
13: s2.prev := s1.prev
```

5.5.2.4 Moving Alt-Join Over Alt-Join (WFT-J4)

In a workflow SWF with a nested alt-structure (i.e. within an or-structure with the split $S1$ and the corresponding join $J1$ there is another alt-structure with the split $S2$ and the corresponding join $J2$), the inner join $J2$ can be moved behind the outer join $J1$, which makes it necessary to move the corresponding split element $S2$. This change causes the inner alt-structure to be put over the outer. An example of this transformation in the workflow SWF' is shown in Fig. 5.10.

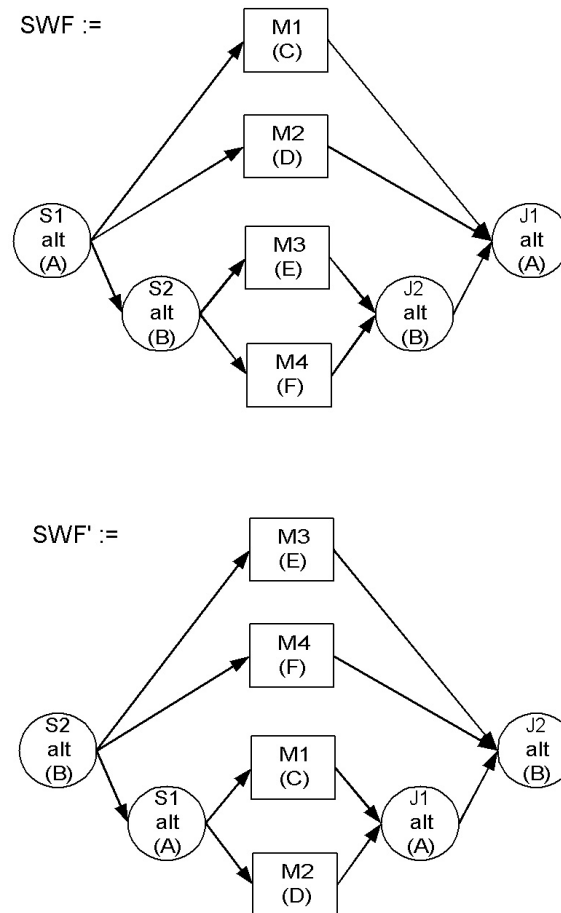
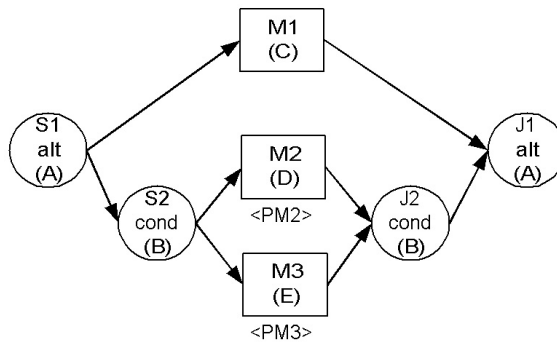


Figure 5.10: Alt-Join Moving Over Alt-Join

5.5.2.5 Moving Or-Join Over Alt-Join (WFT-J5)

In a workflow SWF with a nested alt/or-structure, i.e. within an alt-structure with the split $S1$ and the join $J1$ there is an or-structure with the split $S2$ and the join $J2$, the inner join $J2$ can be moved behind the outer join $J1$. This makes it necessary to move the corresponding split element $S2$ and to duplicate control elements and occurrences and adjust the predicates. This change causes the inner or-structure to be put over the outer. An example of this transformation is given in Fig. 5.11.

$SWF :=$



$SWF' :=$

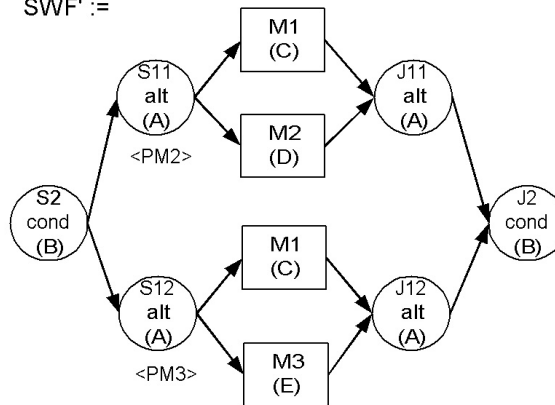


Figure 5.11: Or-Join Moving Over Alt-Join

Algorithm 5.4 describes the transformation operations in pseudocode. The input variable must be an object of the class *ModelControlOccurrence*.

Algorithm 5.4 Or-Join Moving Over Alt-Join

```

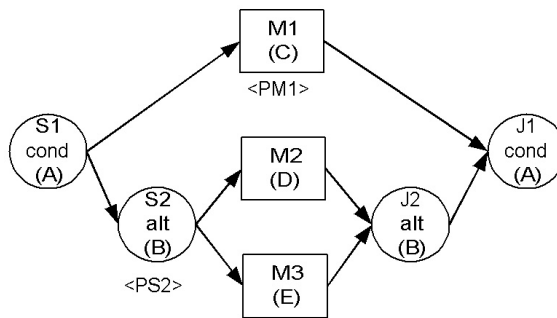
1: Procedure Or-Join Moving Over Alt-Join
2: in:j2:ModelControlOccurrence
3: j1 := j2.next
4: s1 := j1.split
5: s2 := j2.split
6: nj2 := copyModelElement(j2)
7: ns2 := copyModelElement(s2)
8: for all po1 in j2.prev do
9:   nme1j := copyModelElement(j1)
10:  nme1s := copyModelElement(s1)
11:  nme1 := copyModelElement(po1)
12:  insertModelTransition(nme1s, nme1)
13:  insertModelTransition(nme1, nme1j)
14:  for all po2 in (j1.prev - j2) do
15:    nme2 := copyModelElement(po2)
16:    insertModelTransition(nme1s, nme2)
17:    insertModelTransition(nme2, nme1j)
18:  end for
19:  insertModelTransition(ns2, nme1s)
20:  insertModelTransition(nme1j, nj2)
21: end for
22: ns2.prev := s1.prev
23: nj2.next := j1.next
24: removeModelElement(s2.next)
25: removeModelElement(s1.next)
26: removeModelElement(j1.prev)
27: removeModelElement(s1)
28: removeModelElement(j1)

```

5.5.2.6 Moving Alt-Join Over Or-Join (WFT-J6)

In a workflow SWF with a nested or/alt-structure, i.e. within an or-structure with the split $S1$ and the join $J1$ there is an alt-structure with the split $S2$ and the join $J2$, the inner join $J2$ can be moved behind the outer join $J1$. This makes it necessary to move the corresponding split element $S2$ and to duplicate control elements and occurrences and adjust the predicates. This change causes the inner alt-structure to be put over the outer. An example of this transformation is given in Fig. 5.12.

$SWF :=$



$SWF' :=$

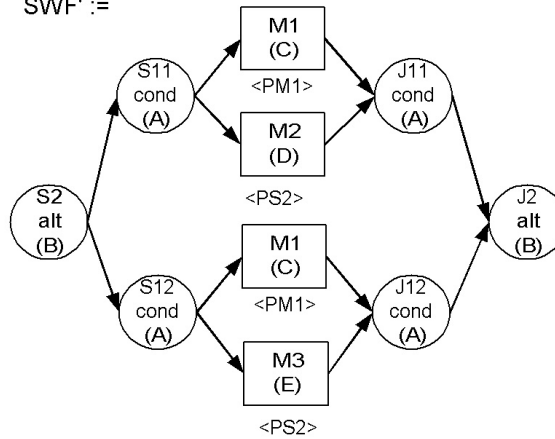


Figure 5.12: Alt-Join Moving Over Or-Join

5.5.2.7 Join Coalescing (WFT-J7)

In a workflow *SWF* with a nested or-structure or alt-structure, i.e. within an or-structure or alt-structure with the split *S1* and the join *J1* there is an or-structure or alt-structure with the split *S2* and the join *J2*, *J2* can be coalesced with *J1*, which makes it necessary to coalesce the corresponding split element *S2* and *S1*. This change causes that the or-structures or alt-structures are replaced by a single one. The predicates must be adapted when or-structures are coalesced. An example for this transformation is given in Fig. 5.13. This transformation is similar to the structurally equivalent transformation presented in [Sadiq and Orłowska, 2000] as far as the differences in the workflow models are concerned.

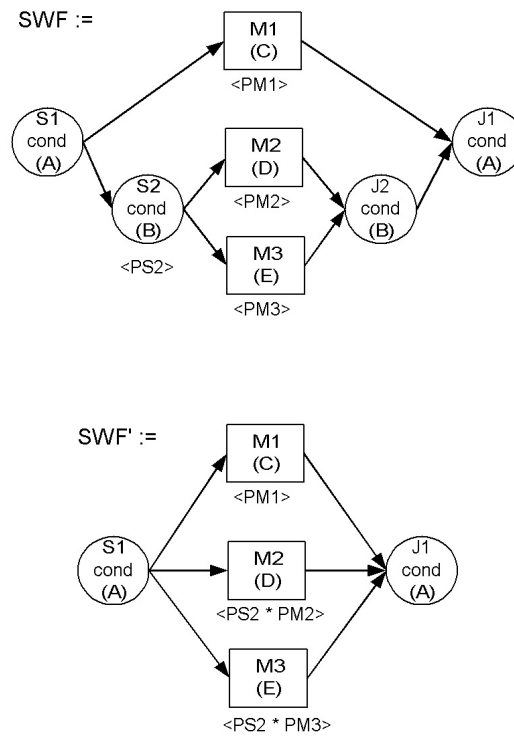


Figure 5.13: Join Coalescing

Algorithm 5.5 describes the transformation operations in pseudocode. The input variable must be an object of the class *ModelControlOccurrence*.

Algorithm 5.5 Join Coalescing

```
1: Procedure Join Coalescing
2:   in:j2:ModelControlOccurrence
3:   j1 := j2.next
4:   s1 := j1.split
5:   s2 := j2.split
6:   nj1 := copyModelElement(j1)
7:   ns1 := copyModelElement(s1)
8:   nj1.prev := (j1.prev  $\cup$  j2.prev) - j2
9:   ns1.next := (s1.next  $\cup$  s2.next) - s2
10:  ns1.prev := s1.prev
11:  nj1.next := j1.next
12:  removeModelElement(s2)
13:  removeModelElement(s1)
14:  removeModelElement(j1)
15:  removeModelElement(j2)
```

5.5.2.8 Separating a Conditional/Alternative Path (WFT-PS)

In a workflow SWF with an or-structure or alt-structure with the split $S1$ and the join $J1$ and a path with node $M1$ (activity or control occurrence), the path with $M1$ can be separated by means of duplicating $J1$, so that the duplicate $J11$ has as predecessor $M1$ and in the set of predecessors of $J1$, the occurrence $M1$ is eliminated. A necessary precondition is that $J1$ has no successor. An example for this transformation is given in Fig. 5.14.

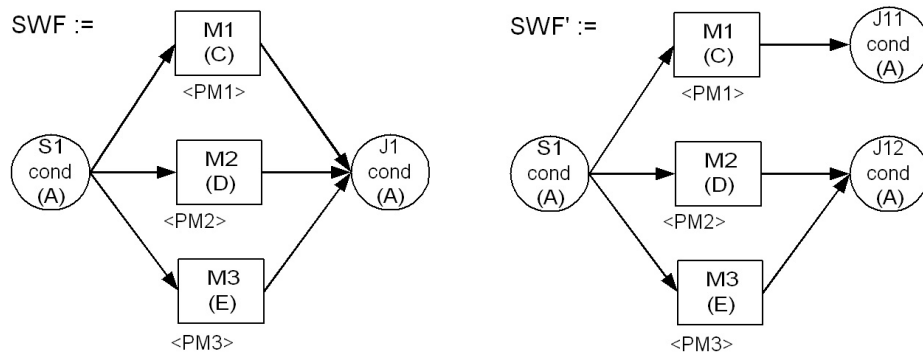


Figure 5.14: Separating a Conditional/Alternative Path

Algorithm 5.6 describes the transformation operations in pseudocode. The input variable must be an object of the class *ModelElement*, which is the predecessor of a join.

Algorithm 5.6 Separating a Conditional/Alternative Path

- 1: Procedure Separate
 - 2: in:m1:ModelElement
 - 3: j1 := m1.next
 - 4: nj1 := copyModelElement(j1)
 - 5: nj1.prev := m1
 - 6: j1.prev := j1.prev - m1
-

5.5.2.9 Moving Join Over And-Join - Unfold (WFT-J8)

The *unfold* transformation produces a graph-based structure which is no longer strictly structured and requires *multiple sequential successors*, which means that a node, with the exception of splits, can have more than one sequential successor in the workflow definition. However, in each instance type every node except for and-splits has only one successor (the other successors of the definition are in other instance types).

An or-join or alt-join *J2* can be moved behind its immediately succeeding and-join *J1*, requiring duplication of control elements. The transformation is shown in Fig. 5.15 and Fig. 5.16. To move *J2* behind *J1* we place a copy of *J1* behind every predecessor of *J2*, so that each of these copies of *J1* has additionally the same predecessor as *J1* except for *J2*. A copy of *J2* is inserted, such that it has the copies of *J1* as predecessor and the successor of *J1* as successor. Then *J1* is deleted with all its successor and predecessor dependencies. If *J2* has no longer a successor, it will also be deleted. *Partial unfold*, as it is described in [Eder et al., 2000], is a combination of already described transformations.

Algorithm 5.7 describes the transformation operations in pseudocode. The input variable must be an object of class *ModelControlOccurrence*.

Algorithm 5.7 Join Moving Over And-Join

```

1: Procedure Unfold
2: in:j2:ModelControlOccurrence
3: j1 := j2.next
4: s1 := j1.split
5: s2 := j2.split
6: for all po2 in j2.prev do
7:   nj1 := copyModelElement(j1)
8:   nj1.pred := (j1.pred - j2) ∪ nj1
9:   po2.next := nj1
10:  nj1.next := j2
11: end for
12: j2.next := j1.next
13: removeModelElement(j1)
  
```

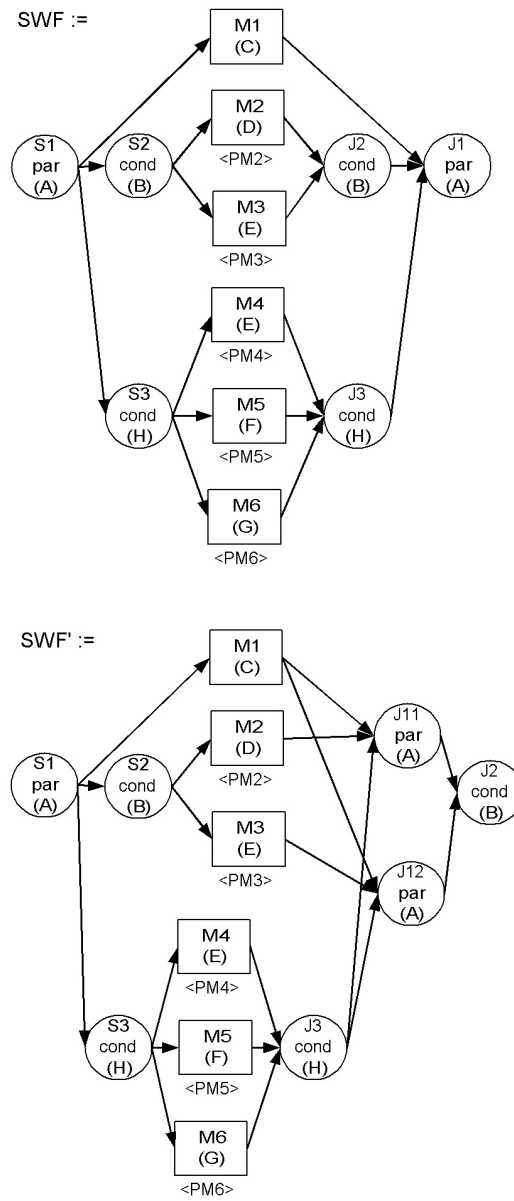


Figure 5.15: Join Moving Over And-Join (Unfold) - 1

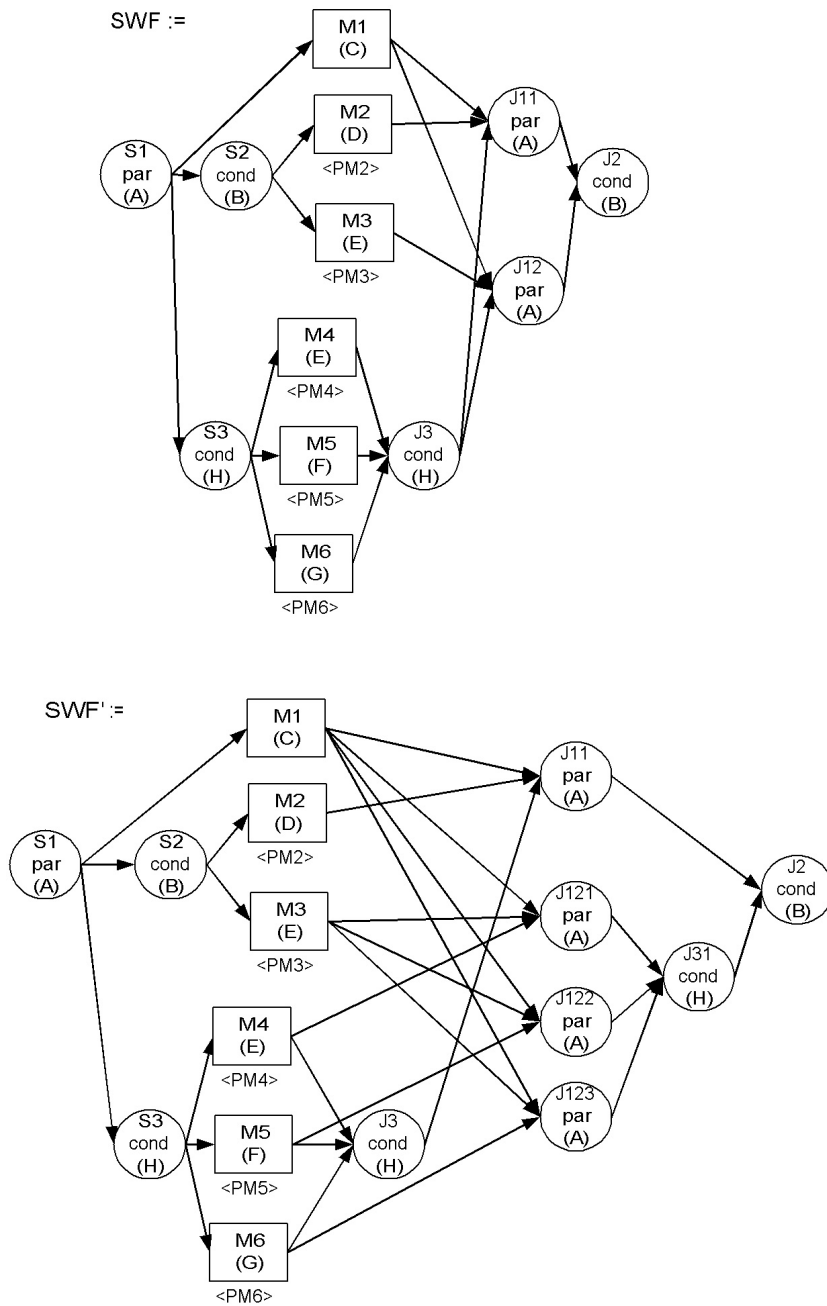


Figure 5.16: Join Moving Over And-Join (Unfold) - 2

5.5.2.10 And-Join Moving Over Or-Join (WFT-J9)

This transformation is introduced in [Kiepuszewski et al., 1999]. Starting with a workflow SWF with an or-join $J1$, which has only and-joins $J2_1 \dots J2_m$ as predecessors, each of these and-joins $J2_i \in \{J2_1 \dots J2_m\}$ has the identical set of predecessors $M_1 \dots M_n$. Let the sets of the predecessors of $M_1 \dots M_n$ for every and-join be $S_1 \dots S_m$. The or-join $J1$ can be moved before the predecessors of the and-joins, which necessitates the duplicating and coalescing of control elements. The transformation is shown in Figure 5.17. In order to move $J1$ we place a copy of $J1$ for every predecessor of an and-joins $J2_i \in \{J2_1 \dots J2_m\}$, so that each copy of $J1$ has the same number of predecessors as $J1$ and every copy of $J1$ has as predecessor one element from every set $S_1 \dots S_m$, so that every element from $S_i \in \{S_1 \dots S_m\}$ has only one successor. Furthermore, we place a copy of an and-join $J2_i$ with its predecessors, so that every copied predecessor of the copied and-join $J2_i$ has exactly one copy of $J1$ as its predecessor. The copy of the and-join $J2$ has as successor the successor of $J1$, if existent. Now, the and-joins $J2_1 \dots J2_m$ with their predecessors and with all their successor and predecessor dependencies are deleted.

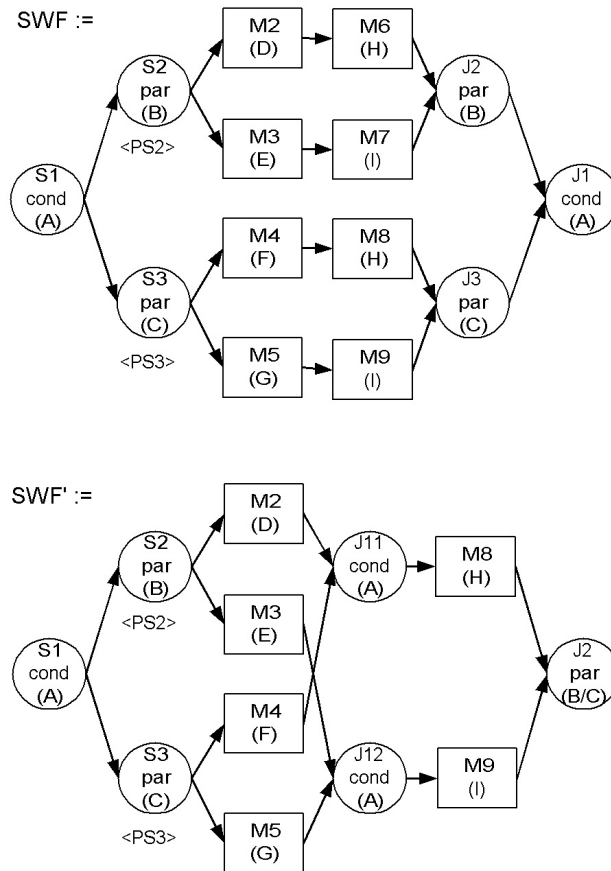


Figure 5.17: And-Join Moving Over Or-Join

Algorithm 5.8 describes the transformation operations in pseudocode. The input variable must be an object of the class *ModelControlOccurrence*.

Algorithm 5.8 And-Join Moving Over Or-Join

```

1: Procedure And-Join Moving Over Or-Join
2: in:j2:ModelControlOccurrence
3: j1 := j2.next
4: nj2 := copyModelElement(j2)
5: for all po2 in j2.prev do
6:   npo2 := copyModelElement(po2)
7:   nj1 := copyModelElement(j1)
8:   npo2.next := nj2
9:   nj1.next := npo2
10: for all po3 in j1.prev do
11:   for all po4 in po3.prev do
12:     if po4.occurrence.activity = npo2.occurrence.activity then
13:       nj1.prev := nj1.prev  $\cup$  po4.prev
14:     end if
15:   end for
16: end for
17: end for
18: for all po3 in j1.prev do
19:   for all po4 in po3.prev do
20:     removeModelElement(po4)
21:   end for
22:   removeModelElement(po3)
23: end for
24: removeModelElement(j1)

```

5.5.3 Split Moving

Split Moving changes the position of a split control element. This transformation separates (moving splits towards start) or merges (moving splits towards end) the intrinsic instance types contained in a workflow model, in analogy to join moving. Not every split can be moved. Moving an alt-split is always possible. For an or-split it is necessary to consider data dependencies on the predicates. Another aspect of or-split moving to be considered is that the decision which path of an or-split is selected will be transferred forward, so that uncertainty based on or-splits will be reduced.

5.5.3.1 Moving Split Before Activity Occurrence (WFT-S1)

A workflow *SWF* with an or-split or alt-split *S1* with activity occurrence *M1* as predecessor can be transformed in the workflow *SWF'* through node duplication, so that *S1* is located before *M1* (see Fig. 5.18). Here, *M1* will be replaced by its duplicates *M11* and *M12*, so that *S1* is the predecessor of *M11* and *M12*, and *M2* is the successor of *M11* and *M3* is the successor of *M12*. Predicates are adjusted.

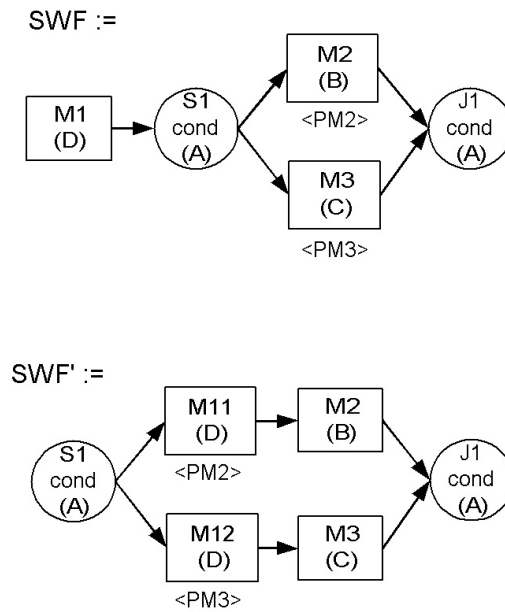


Figure 5.18: Split Moving Before Activity

Algorithm 5.9 describes the transformation operations in pseudocode. Here, the method *copyModelElement(in)* creates a copy of the object *in*. The remaining methods are described above. The input variable must be an object of the class *ModelControlOccurrence*.

Algorithm 5.9 Split Moving Before Activity

```

1: Procedure Split Moving Before Activity
2: in: s1:ModelControlOccurrence
3: for all po in s1.next do
4:   nme := copyModelElement(s1.prev)
5:   insertModelTransition(s1, nme)
6:   insertModelTransition(nme, po)
7:   removeModelTransition(s1, po)
8: end for
9: te := s1.prev
10: s1.prev := te.next
11: removeModelElement(te)

```

5.5.3.2 Split Moving Over Seq-Join (WFT-S2)

A workflow SWF with an or-split or alt-split $S1$ preceded by a sequence split $S2$ can be transformed to workflow SWF' through node duplication, so that the split $S2$ is delayed after $S1$ as shown in Fig. 5.19. Here, $S2$ will be replaced by its duplicates $S21$ and $S22$, so that $S1$ is the predecessor of $S21$ and $S22$, and $M1$ is the successor of $S21$ and $M2$ is the successor of $S22$. This transformation results in a less structured workflow.

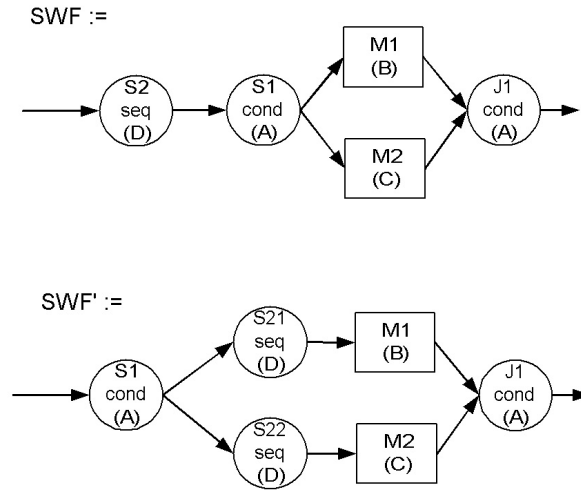


Figure 5.19: Split Moving Over Seq-Split

5.6 Complex Transformations

The basic workflow transformations above can be used to define more complex transformations as compositions with facultatively repeated application of these basic transformations within their composition. These complex transformations do not change the semantics of the workflow either.

Complex transformations are established to e.g. address the time constraint incorporation problems as stated in [Eder et al., 2000]. In that paper, three complex transformations are constructed: (1) the *backward unfolding procedure*, (2) the *partial backward unfolding procedure*, and (3) the *forward unfolding procedure*. Unfolding means that or-joins or alt-joins are moved topologically to the rear and or-splits or alt-splits are moved as near to the start as possible and thus the intrinsic instance types are separated.

In the following Sections, these transformations are outlined verbally, graphically and with pseudocode. Table 5.2 gives a short overview of the complex workflow transformations.

Table of Complex Workflow Transformations	
Name	Description
CWFT1	Backward Unfolding Procedure
CWFT2	Partial Backward Unfolding Procedure
CWFT3	Forward Unfolding Procedure

Table 5.2: Overview of Complex Workflow Transformations

5.6.1 Backward Unfolding Procedure

A procedure for generating an equivalent backward unfolded workflow UW for a workflow W is described in [Eder et al., 2000]. The transformation specifies how a workflow has to be modified to become fully unfolded. An alternative approach to unfold a workflow is to apply the above listed basic transformations in a way that no or-join or alt-join element has an activity as successor. Every (less) structured workflow can be fully unfolded, because for every constellation there is a basic transformation that can be applied in order to move the corresponding join topologically backward. The following constellations are identified:

- or-/alt-join before activity - use WFT-J1
- or-/alt-join before and-join - use WFT-J8
- or-/alt-join before seq-join - use WFT-J2
- or-/alt-join before or-/alt-join - use WFT-J3, WFT-J4, WFT-J5, or WFT-J6
- separate path - use WFT-PS

Backward unfolding is applied in the following workflow example. Figure 5.20 represents the initial workflow. The or-joins *M9* and *M18* have to be moved to the rear.

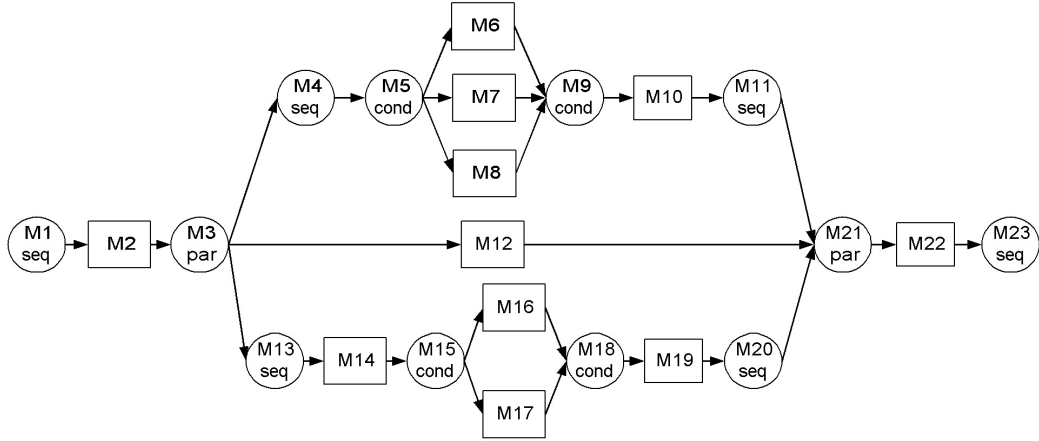


Figure 5.20: Aggregated Workflow

Figure 5.21 shows an intermediate step of the transformation, where *M9* and *M18* are moved over the adjacent activity occurrence *M10* or *M19*.

Figure 5.22 shows the unfolded workflow for the workflow shown in Figure 5.20.

Algorithm 5.10 describes the fully unfold transformation operations in pseudocode. Here, the method *apply(wf, set_of_transformations)* returns *true*, if a transformation operation in *set_of_transformations* can be applied on *wf*, otherwise it returns *false*.

Algorithm 5.10 Backward Unfolding Procedure

- 1: Procedure Backward Unfold
 - 2: in: wf:Workflow
 - 3: sot :=WFT-J1, WFT-J2, WFT-J3, WFT-J4, WFT-J5, WFT-J6, WFT-J8,WFT-PS
 - 4: **while** wf is not fully unfolded **do**
 - 5: apply(wf,sot)
 - 6: **end while**
-

The above procedure suffers from the potential explosion of the number of “duplicate” nodes in the unfolded workflow, since it considers each instance type

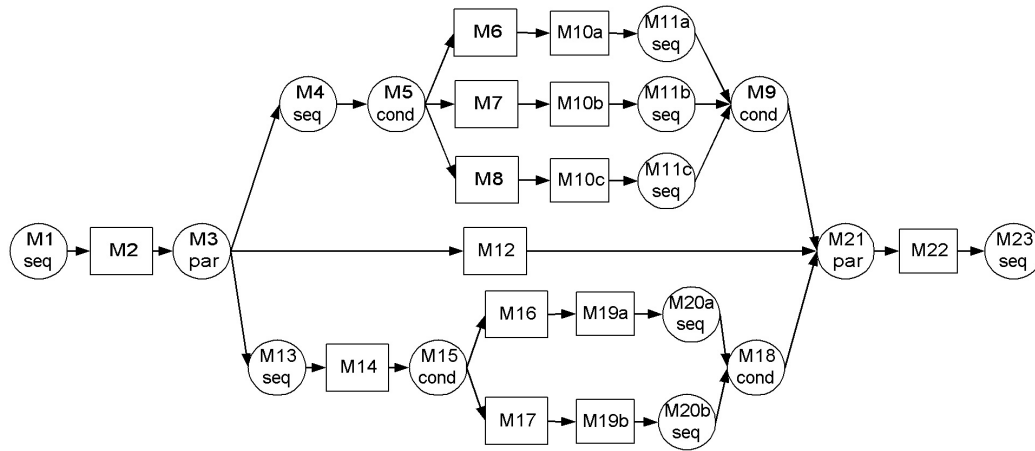


Figure 5.21: Prepared Workflow

separately. This is not always desirable when discriminating between instance types. To avoid this problem, we developed the *partial unfolding* technique (see [Eder et al., 2000]).

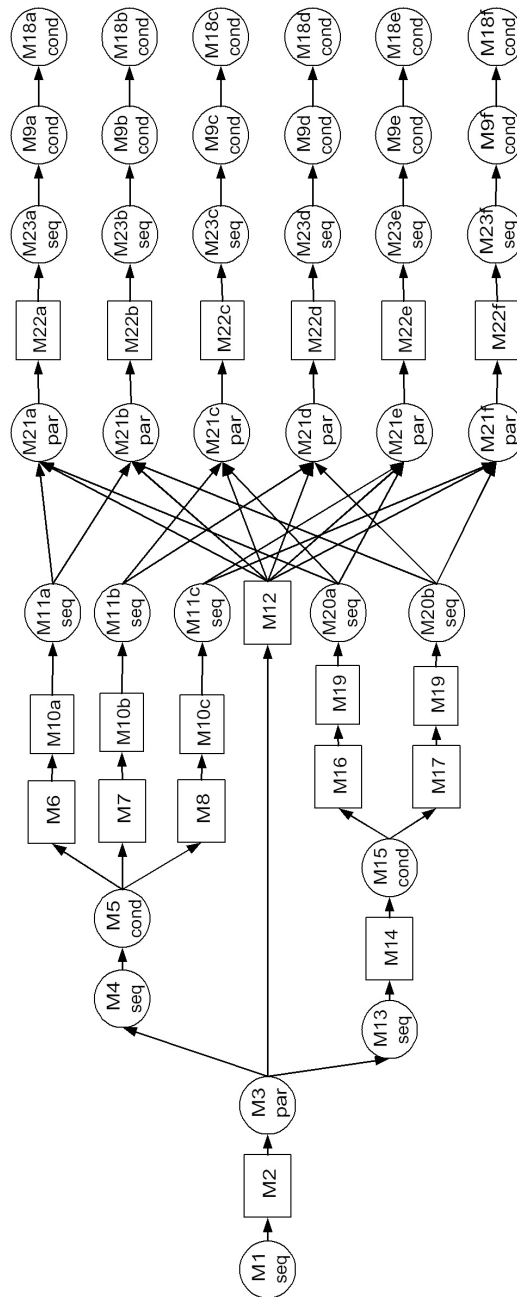


Figure 5.22: Unfolded Workflow

5.6.2 Partial Backward Unfolding Procedure

We can unfold the workflow only where it is desired. The procedure of partially unfolding a workflow G to a workflow H begins by selecting a *hot-node*, with the side effect that all instance types going through the hot-node are factored out, or intuitively, the workflow graph reachable from the hot-node is duplicated [Eder et al., 2000]. In principle, every node can be a hot-node. For practical reasons, we assume that a hot-node is an immediate predecessor of an or-join. In the next Chapter we will show how hot-nodes are chosen when a time constraint cannot be incorporated. Once a hot-node is identified, partial unfolding takes place as follows:

- 1 Mark all (transitive) successors of the hot-node;
- 2 Apply the transformations *WFT-J1*, *WFT-J2*, *WFT-J3*, *WFT-J4*, *WFT-J5*, *WFT-J6*, *WFT-J8*, *WFT-PS* on the marked workflow elements so that no or-join or alt-join element has an activity element as successor in the context of the marked workflow elements;

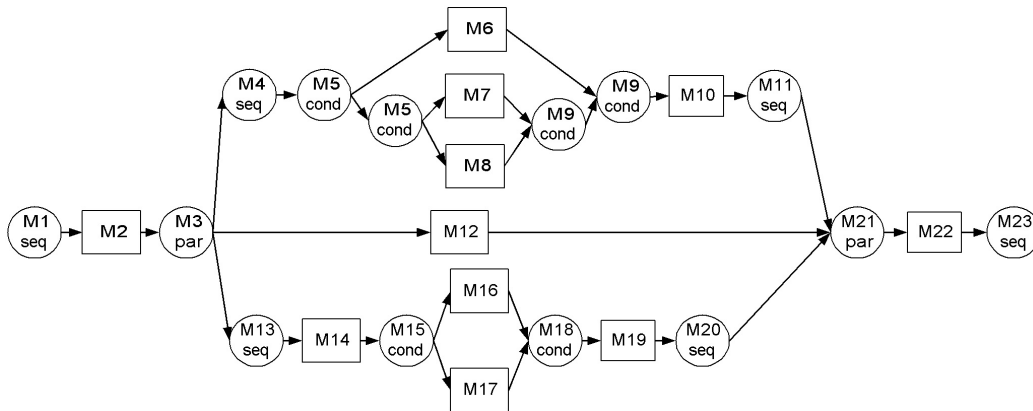


Figure 5.23: Intermediate Partially Unfolded Workflow

Based on this procedure, we partially backwardly unfold the workflow shown in Figure 5.20 at node $M6$. Figure 5.23 shows the prepared workflow, and Figure 5.24 shows the resulting partially unfolded workflow.

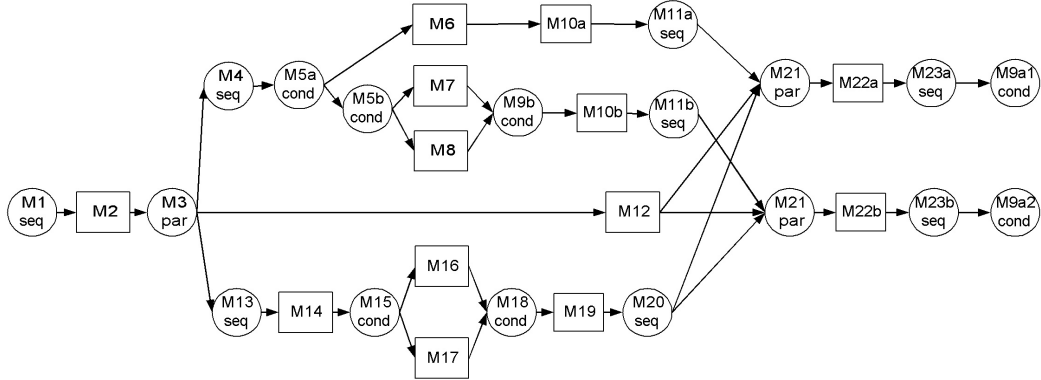


Figure 5.24: Partially Unfolded Workflow

Algorithm 5.11 describes the partial unfold transformation operations in pseudocode. Here, the method *mark_successors(hot-node)* marks all (transitive) successors of the hot-node and returns the marked elements as a (*sub*-)workflow .

Algorithm 5.11 Partial Backward Unfold

- 1: Procedure Partial Backward Unfold
 - 2: in: hot-node:Workflow
 - 3: sot := WFT-J1, WFT-J2, WFT-J3, WFT-J4, WFT-J5, WFT-J6, WFT-J8
 - 4: sub_wf := mark_successors(hot-node)
 - 5: **while** sub_wf is not fully unfolded **do**
 - 6: apply(sub_wf,sot)
 - 7: **end while**
-

5.6.3 Regarding Transformation Sequence When Unfolding

When (partially) unfolding a workflow, the sequence of applying the transformation operations is of great importance. As it is shown below, it is sometimes necessary that transformation steps are rolled back in order to achieve the goal, which is especially true for less structured workflows. In this context it is worth considering the workflow depicted in Figure 5.25, where we want to unfold the workflow affecting the path with occurrences *A* and *C*.

The first transformation step is to unfold the workflow concerning occurrence *C*. The result is shown in Figure 5.26. Here, we cannot apply any further unfold transformation because of the less strict structure that results in overlapping structures.

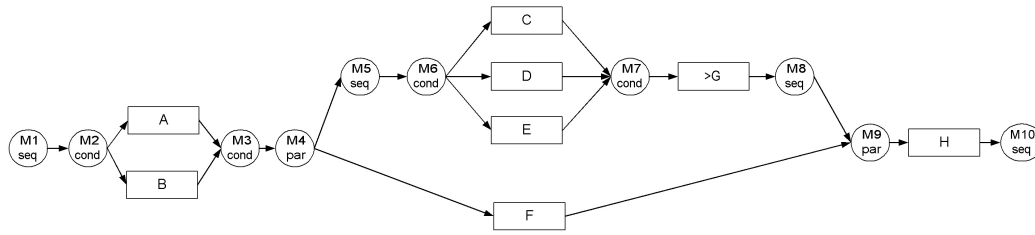


Figure 5.25: Unfold Workflow Procedure

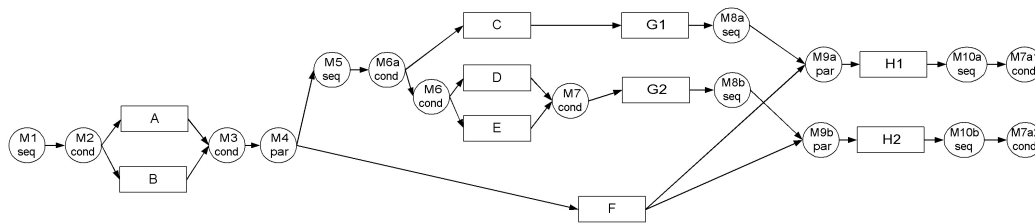


Figure 5.26: Unfold Workflow Procedure - 1

Therefore, we withdraw the first step and unfold the workflow regarding occurrence A, which is depicted in Figure 5.27 (occurrences X1 and X2 represent the complex occurrence belonging the parallel structure bound by M4 and M9).

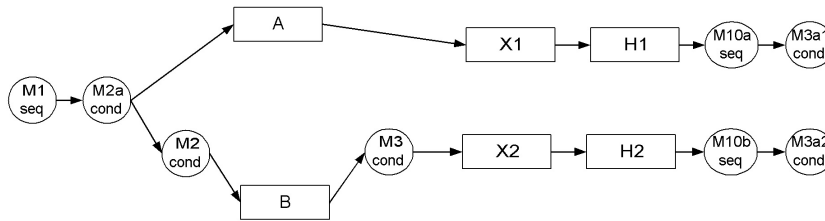


Figure 5.27: Unfold Workflow Procedure - 2

After that transformation step we separate the path with occurrence C. The result is shown in Figure 5.28.

To sum up, the transformation step order is of great importance to attain the workflow aimed for and to avoid unnecessary cancellations of operations.

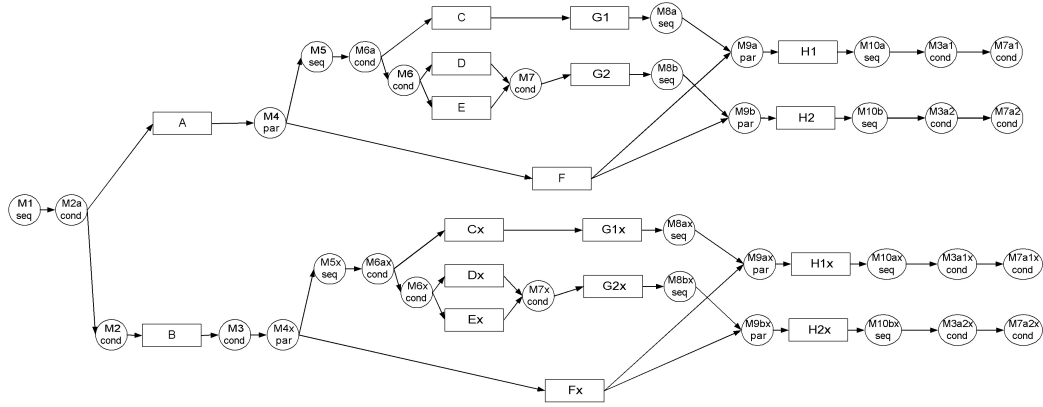


Figure 5.28: Unfold Workflow Procedure - 3

5.6.4 Forward Unfold Procedure

A procedure for generating an equivalent forward unfolded workflow UW for a workflow W is described below. In order to unfold a workflow forward the transformations listed above must be applied so that no or-split and no alt-split element has an activity element as predecessor. Because of data dependencies, thought, not every (less) structured workflow can be fully forward unfolded.

For the following constellations there is a basic transformation that can be applied in order to move the corresponding split topologically forward. The following constellations are identified:

- or-/alt-split after activity without data dependencies - use WFT-S1
- or-/alt-split after seq-split - use WFT-S2
- or-/alt-split after or/alt-join - use WFT-J3, WFT-J4, WFT-J5, or WFT-J6

For the constellations *or-/alt-split after and-join* there does not exist a transformation.

Algorithm 5.12 describes the fully forward unfold transformation operations in pseudocode. Here, the method *apply(wf, set_of_transformations)* returns *true* if any transformation operation in *set_of_transformations* can be applied on *wf*. Otherwise it returns *false*.

Algorithm 5.12 Forward Unfolding Procedure

```
1: Procedure Forward Unfold
2: in: wf:Workflow
3: sot :=WFT-S1, WFT-JS2, WFT-J3, WFT-J4, WFT-J5, WFT-J6
4: while wf is not fully unfolded and apply(wf,sot) do
5:
6: end while
```

5.7 Application

There are several applications for the presented methodology. Thus, it serves as sound basis for design tools (see Chapter 7). Furthermore, it enables analysts and designers to incrementally improve the quality of the model. We can provide automatic support to achieve certain presentation characteristics of a workflow model. A model can be transformed to be inspected it from different points of view. In particular a model suitable for conceptual comprehension can be transformed to a model better suited for implementation.

5.8 Summary

This Chapter presents an approach to tackle the problem of change without modifying the semantics. The main contribution is the development of a set of basic workflow scheme transformations. Based on it, several complex transformation operations that maintain the semantics are also defined. In order to accomplish transformations preserving equivalence, we introduced a new equivalence criterion on workflows.

6

Time Management in Workflows

In this Chapter, we address the crucial role of time management in workflow processes. In particular, we describe how structural (*i.e.*, execution order dependent) and explicit (*i.e.*, fixed-date, periodic, upper- and lower-bound) time constraints can be “captured” during process definition and validated during modeling time. Therefore, the basic temporal concepts used in workflows are introduced and a framework for time modeling is presented.

This Chapter is organized as follows: Section 6.1, generally introduces the particular importance of time management in enterprises. Section 6.2 gives an overview of related work in literature. In Section 6.3, time aspects within the scope of WfMSs are discussed. Sections 6.4 and 6.5 deal with time constraints and our time model. Sections 6.6 and 6.7 show our corresponding graphical model for workflows. Sections 6.8, 6.9 and 6.10 provide the methods and algorithms for our time management framework. A short example of how to apply our techniques is given in Section 6.11. Finally, Section 6.12 concludes this Chapter.

6.1 Introduction

Today, the most critical requirement of companies striving to become more competitive is the ability to control the flow of information and work throughout the enterprise in terms of time. Workflow management systems (WfMSs) improve business processes by automating tasks, by getting the right information to the right place for a specific job function, and by integrating information in the enterprise [Lawrence, 1997; Georgakopoulos et al., 1995; WfMC, 1999a; Hollingsworth, 1995]. However, existing WfMSs [InC; Leymann and Roller; Flo] offer only limited support for modeling and managing time constraints associated with processes and their activities [Bettini et al., 2002; Pozewaunig et al., 1997]. This support takes place through monitoring activity deadlines [Schmidt, 1996].

However, the consistency of these deadlines and the side effects of not meeting them are not addressed [Eder et al., 2000].

In process centered organizations, time management is essential for process modeling and management. Many business processes have restrictions such as limited duration of subprocesses, terms of delivery, dates of re-submission, or activity deadlines. Generally, time violations increase the cost of a business process because they lead to some form of exception handling [Panagos and Rabinovich, 1997b]. Therefore, a WfMS should provide the necessary information about a process, its time restrictions, and its actual time requirements to a process manager. In addition, the process manager needs tools to anticipate time problems, to avoid time constraints violations proactively, and to make decisions about the relative priorities of processes and timing constraints [Eder et al., 2000].

In the following, the concepts of time management, introduced in [Eder et al., 2000 1999b; Pozewaunig et al., 1997], and the workflow meta model of Chapter 4, introduced in [Eder and Gruber, 2002] are brought together resulting in an extended meta model. The notion of *timed workflow graphs* that was introduced in [Eder et al., 1999a; Pozewaunig et al., 1997] is used to show how time information is represented in these graphs. In [Eder et al., 2000 1999b], where *explicit temporal constraints* and a technique for incorporating these constraints into timed workflow graphs was introduced, is also made use of for our new concepts.

6.2 Related Work

A large amount of work has been done on workflow systems across the fields of CSCW (Computer Supported Cooperative Work) and advanced transaction models. Functionality and limitations of workflow systems have been analyzed in several papers (see e.g. [Alonso et al., 1997]).

Although the field of time management has received a lot of attention in areas such as project management, job-shop scheduling, and active databases, currently available commercial workflow products provide little support beyond simply monitoring activity deadlines. Commercial workflow systems (as reviewed, e.g., in [Alonso et al., 1997]) are usually limited to the specification of a deadline for each activity or global plan. In some cases more elaborate temporal conditions can be specified, but no reasoning other than run-time evaluation on these conditions is supported. Furthermore, workflow research has recently started addressing time management issues [Eder and Panagos, 2000; Eder et al., 2000]. Up to now, however, little attention has been paid to modeling advanced temporal features for workflow systems, except for some exceptions [Bettini et al., 2002].

A good overview of time management literature can be found in [Bettini et al., 2002; Eder and Panagos, 2000], which is largely reproduced below and complemented with current work in this field.

An ontology of time for identifying time structures in workflow management systems is developed in [Jasper and Zukunft, 1996]. The authors represent time aspects within a workflow environment by using the Event Condition Action (ECA) model found in active database management systems. Furthermore, they discuss special scheduling aspects and basic time-failures. The setting of internal deadlines differs from dynamic workflow modification that is supported by some existing workflow products and research prototypes. The latter is done to reflect changes in the model of the business process or a particular instance of the process. In contrast, our goal is to capture time information at build time and to cope with explicit time constraints.

In [Kao and Garcia-Molina, 1993ab], the authors examined the problem of how the deadline of a real-time activity is automatically translated to deadlines for all sequential and parallel sub-tasks constituting the activity. Each sub-task deadline is assigned just before the sub-task is submitted for execution, and the algorithms for deadline setting assume that the *earliest deadline first* scheduling policy is used. While our work is partly similar to the above work, there are several important differences. In particular, we treat alternative and conditional activities. Also, we offer techniques for building the timed graph at process build time. Moreover, our work supports the assignment of external deadlines to individual activities as well as to the entire process.

In [van der Aalst et al., 1994], the authors show how high-level Petri nets are used to model workflows in an office environment. Such nets, which are extended with “color”, “time” and “hierarchy”, describe i.e. the temporal behavior of real systems. However, temporal reasoning could not be performed by means of high-level Petri nets.

[Panagos and Rabinovich, 1996 1997ab] suggest the use of static data (e.g. escalation costs), statistical data (e.g. average activity execution time and probability of executing a conditional activity), and run-time information (e.g. agent work-list length) to adjust activity deadlines and estimate the remaining execution time of workflow instances. However, this work can be used only at run-time and, furthermore, it does not address explicit time constraints.

In [Bussler, 1998], the author proposes the integration of workflow systems with project management tools to provide the functionality necessary for time management. However, these project management tools do not allow the modeling of explicit time constraints and, therefore, they do not provide any means for their resolution.

In [Pozewaunig et al., 1997], the authors present an extension to the net-diagram technique PERT to compute internal activity deadlines in the presence of sequential, alternative, and concurrent executions of activities. Using this technique, business analysts provide estimates of the best, worst, and median execu-

tion times for activities. The β -distribution is used to compute activity execution times as well as the shortest and longest process execution times. Having done that escalations are monitored at run-time based on the ePERT. Our work extends this work by handling both structural and explicit time constraints at process build and instantiation times.

In [Eder et al., 1997a; Pozewaunig, 1996], the notion of explicit time constraints is introduced. Nevertheless, this work focused more on the formulation of time constraints, the enforcement of time constraints at run-time and the escalation of time failures within workflow transactions [Eder and Liebhart, 1997]. Our work follows the work described in [Eder et al., 1997a; Pozewaunig, 1996] and extends it with the incorporation of explicit time constraints into workflow schedules.

In [Dadam et al., 2000], the authors describe some of the time-related functionalities of the *ADEPT_{time}* workflow management system. As part of the time functionality, minimal and maximal durations may be specified for each workflow activity. In addition, time dependencies between workflow activities may be defined. These dependencies are the same as the lower bound and upper bound constraints we presented in this thesis, and they are modeled using an additional edge that links the activities involved in such constraints. At build time, the existence of a valid time schedule is checked (i.e. an assignment of absolute start and finish times so that all constraints are satisfied). Start and finish times for activities are calculated at run-time using the Floyd-Warshall algorithm, and users are notified when deadlines are going to be missed.

Temporal constraint networks in the context of workflows have already been dealt with in [Haimowitz et al., 1996], who were concerned with health care enterprises, and in [Zhao and Stohr, 1999], who examined a claim handling system to track individual tasks subject to constraints with other tasks in the workflow. However, both papers do not address the problem of explicit time constraints.

[Casati et al., 1999] describe the WIDE system developed in the context of a European project on workflow systems. In that chapter of the book, the definition of temporal information is considered to be of great importance for these systems, particularly for handling temporal exceptions. Primitives are introduced to model time intervals, durations and periodic conditions, but there has been no investigation about temporal constraints in this domain.

The work presented in [Marjanovic, 2001 2000; Marjanovic and Orlowska, 1999ba] is close to our work. The authors propose a framework for time modeling in production workflows. They apply the same constructs for workflow specification and similar temporal constraints. Duration constraints in the form [min, max] for each activity involved are an integral part of a workflow specification. They provide an algorithm to check that any other type of constraint that may be

needed (deadline and inter-task constraints) is implied by the duration constraints and the workflow structure. This is similar to how checking is performed in our framework. However, there are important differences between the two of them. In contrast to the above mentioned work, we do not consider time constraints in isolation and provide solutions for overlapping, interleaving, and interfering constraints. Furthermore, between the activities that belong to a time constraint, there must be a path which is unnecessary in our approach. As demonstrated in [Eder et al., 1999b], a set of time constraints may be unsolvable (i.e. there is no instance of a workflow that does not violate at least one time constraint) even if every single constraint is solvable in isolation. In addition, our techniques are pro-active in nature, and they attempt to modify the E- and L-values of activities to make constraints satisfying.

In [Zhuge et al., 2001 2000], the authors propose a framework for time modeling in workflows. They consider (i) the same constructs for workflow specification as we do, (ii) multiple time axes, and (iii) transition durations. However, transition durations can be modeled by a transition task or multiple alternative transition tasks. There are some shortcomings: they do not treat and-structures and or-structures differently when time information is calculated; explicit time constraints are not taken into consideration. Inconsistency can be found on the build-time calculation, where run-time terms are used.

In [Bettini et al., 2002 2000] the authors propose to enhance the capabilities of workflow systems to specify quantitative temporal constraints on the duration of activities and synchronization requirements. In particular, they investigate consistency (to ensure that the specification of the temporal constraints is possible to satisfy), prediction (to foretell the time frame for the involved activities), and enactment services (to schedule the activities so that, as long as each agent starts and finishes its task within the specified time period, the overall constraints will always be satisfied) in a workflow system by providing corresponding algorithms. The authors also generate schedules considering different time granularities. In contrast to [Bettini et al., 2002], we do not check workflow instance types for constraint violation in isolation. As demonstrated in Section 6.10.1, this is not sufficient and may lead to impossible schedules with contradictory intervals.

This thesis adapts and extends the time modeling and management technique presented in [Eder et al., 2000 1999b] and puts it together with the work in [Eder et al., 1999a; Pozewaunig et al., 1997]. In particular, we extend the expressiveness of the workflow model by augmenting it with procedures for dealing with conditional executions (see [Eder et al., 2000]) and alternative executions. Consequently we have to adapt the computation of timed graphs and the incorporation algorithm for explicit time constraints to the increased expressiveness and complexity of execution constructs.

6.3 Time Modeling in Workflows

It is imperative that current and future WfMSs provide the necessary information about a process, its time restrictions, and its actual time requirements to process modelers and managers. This information can be represented by a workflow process definition.

We basically distinguish between three categories of time periods considering workflows process definitions:

- 1 **Build-Time** The time period when manual and/or automated (workflow) descriptions of a process are defined and/or modified electronically [WfMC, 1999a].
- 2 **Run-Time** The time period when workflow processes are instantiated (enactment) and executed.
- 3 **Post-Run-Time** The time period after instantiated workflow processes are terminated.

These time periods are ordered timely and may overlap, and they are responsible for different tasks, as can be seen in Figure 6.1.

6.3.1 Build-Time, Run-Time and Post-Run-Time

At *build-time*, when workflow schemes are defined and developed, workflow modelers need means of representing time-related aspects of business processes (activity durations, time constraints between activities, *etc.*) and of checking their feasibility [Eder and Panagos, 2000].

At *run-time*, when workflow instances are instantiated and their executions are started, process managers should be able to monitor and adjust time plans (*e.g.*, extend deadlines) according to time constraints and any unexpected delays. Furthermore, they need pro-active mechanisms for being notified about possible time constraint violations so that they can take the necessary steps to avoid time failures. Workflow participants need information about urgencies of the tasks assigned to them to manage their personal work lists. If a time constraint is violated, the WfMS should be able to trigger exception handling to regain a consistent state of the workflow instance [Eder and Panagos, 2000].

At *post-run-time*, when workflow instances are terminated and their executions are finished, business process re-engineers get information about the actual time consumption of workflow executions via workflow documentation (also referred to as *workflow history* or *workflow logging*) [Eder et al., 2002; Geppert and Tombros, 1997] and monitoring interfaces to improve business processes. Finally, controllers and quality managers need information about activity start times and execution durations. Figure 6.1 shows the time modeling process model as described above.

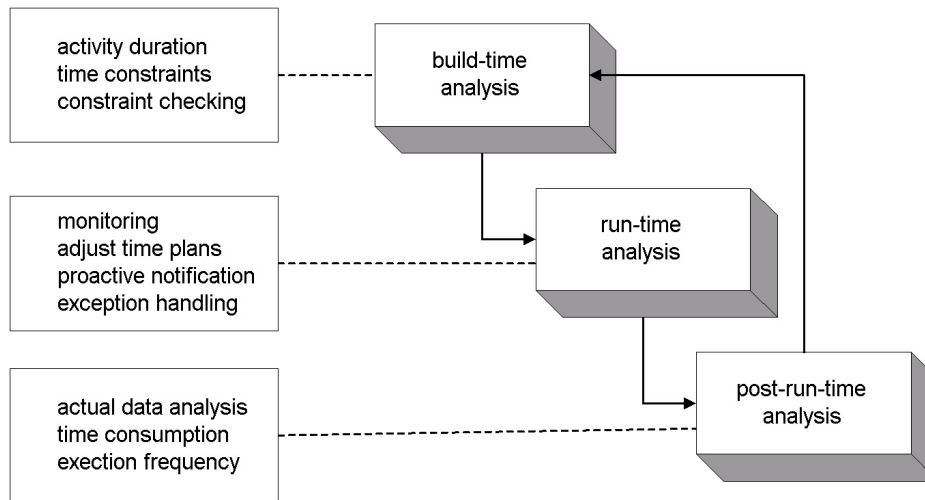


Figure 6.1: Process Model for Time Management in WfMSs

In this Chapter, we are mainly interested in the following aspects at build-time:

- Modeling of time and time constraints to capture the available time information;
- Pro-active time calculations to capture time constraint violations and raise alerts in case of potential future time violations;
- In the case of time constraint violations we are deforming (transforming) the workflow without any semantic change so that superfluous constraint violations are avoided.

We should note, however, that the effectiveness of time management depends on the kind of the workflow, how detailed its description is, and whether there are external causes for time relevant events. For highly structured, production-based workflows, time requirements can be calculated with a high degree of accuracy. For administrative workflows, which cover different organizations, which depend on external events (e.g. waiting for a customer to reply), or which may change their scheme during execution, time calculations are imprecise. Nevertheless, time management, time planning, and controlling has to be done, and, according to our experience, it is being done in current business processes. Usually, time planning relies on estimates based on experience. Time management during the execution of a process becomes even more important in such an environment, where time monitoring is essential for adjusting plans to avoid deadline violations. Therefore, any knowledge about time issues should be modeled and used during workflow execution [Eder and Panagos, 2000].

6.3.2 Time Aspects in Workflow Management Systems

In several application domains of WfMSs the supervision of firm deadlines is inevitable, e.g. for the medical treatment of patients in a hospital environment or for the assembly of products with an ensured delivery date. The same holds true for software engineering projects, especially when they follow a predefined process model [Jasper et al., 1996].

6.3.2.1 Ontology of Time

In our time ontology, we choose a discrete model of time¹ in which time is viewed as isomorphic to the integers [Clifford and Rao, 1987].

Conceptually, time is linear and time points (chronons) can be identified with the integers under the usual ordering $<$. The difference of any two time points is likewise an integer. An interval is a pair of points $< n, m >$, where $n < m$ [Bettini et al., 1998; Kautz and Ladkin, 1991].

Behavior over time is represented in terms of events and links between the events. An event e represents instantaneous changes of qualitative values of parameters and their resultant values at a time point. Changes of quantitative values are assumed to be continuous and differentiable (cp. [Bettini et al., 2002]).

However, time is expressed in some basic time units relative to a time origin which is usually the start of the workflow.

6.3.2.2 Execution Durations and Deadlines

Activity and process deadlines correspond to maximum allowable execution times for activities and processes, respectively. At process build time, these deadlines are specified relative to the beginning of the process, using some time granularity, e.g. 2 hours, 5 minutes, or by Wednesday. At process instantiation time, a calendar is used to convert all relative deadlines to absolute time points, to modify the assigned deadlines, or to assign new deadlines [Eder and Panagos, 2000].

It is important to note that activity durations and deadlines may not be the same, which is, however, the way they are regarded as by some of the existing workflow management systems. To distinguish between activity duration and deadline is beneficial for cases where the actions taken when a deadline is missed might lead to high costs (e.g. rollback of the entire process). In cases, where an activity takes longer to be executed than the duration assigned to it in the workflow scheme, preemptive steps can be taken to assess deadline satisfiability, to modify workflow parameters, and to alert appropriate agents and process managers [Eder and Panagos, 2000; Eder et al., 1999b].

¹ There are also the continuous model (isomorphic to the real numbers) and the dense model (isomorphic to the rationals).

Deadlines do not have to be associated with every activity occurrence of a workflow scheme. However, it is highly beneficial to assign deadlines to all activities. The most compelling reason for this is the ability to monitor the execution progress of activities and processes, so that preemptive actions are taken when delays are developed. We present how these deadlines, referred to as *internal deadlines*, are computed at process build time [Eder and Panagos, 2000].

6.4 Time Constraints

The temporal constraints used in this thesis can be seen as a generalized version of gap-order constraints as introduced in [Revesz, 1993]. In particular, we allow both a minimal and a maximum bound on the distance between two variables. The notion of temporal constraint network used in this thesis is deeply investigated in [Schwalb and Vila, 1998; Dechter et al., 1991].

Time constraints (temporal constraints) are different rules that regulate the time component of a workflow [Marjanovic, 2000]. These constraints are derived implicitly from control dependencies and activity durations on the one hand, and from organizational rules, laws, commitments, etc. on the other hand (e.g. an appeal can be filed within 7 days after the verdict, a meeting invitation has to be sent to all participants at least one week before the meeting) [Eder and Panagos, 2000; Eder et al., 2000 1999b]. Hence, we can identify two categories of time constraints in general: *implicit time constraints* and *explicit time constraints*.

6.4.1 Implicit Time Constraints

Given a workflow scheme, a workflow designer can assign execution durations to individual elementary activities [InC; Leymann and Roller; Flo] at the specification level. These durations can be either calculated from past executions (workflow logs), or they can be assigned by specialists based on their experience and expectations. In addition, multiple execution durations may be assigned to some activity (cf. [Eder and Pichler, 2002]). Usually, the most common duration values used include minimum, maximum, and most frequent execution times. Here, we take the average duration.

Many time constraints are derived implicitly from control dependencies between occurrences of activities at the model level. They arise from the fact that an activity occurrence can only start when its predecessor activity occurrences have finished. Such constraints are called *structural time constraints* because they reflect the control structure of the workflow.

6.4.2 Explicit Time Constraints

In addition, workflow designers may specify *explicit time constraints*. Such explicit constraints are either (1) temporal relations between start and end of (different) occurrences (events), or (2) bindings of events to certain sets of calendar dates [Eder et al., 1999b; Eder and Panagos, 2000]. As mentioned above, these constraints are based on organizational rules and business policies, laws and regulations, service-level agreements, commitments, and so on. Examples of such constraints include: (1) an invitation for a meeting has to be mailed to the participants at least one week before the meeting; (2) after a hardware failure is reported, a service team should be at the customer's site within 4 hours; (3) vacant positions can be announced on the first Wednesday of each month; (4) inventory checks should be finished by December, 31st; (5) loans above 1 mio. € are approved during scheduled meetings of the board of directors.

6.4.2.1 Temporal Relationship Constraints

For temporal relationships between events the following time constraints can be defined, assuming that δ corresponds to a relative time duration that is expressed in some time granularity (cf. [Eder et al., 1999b]).

- **Lower bound constraint:** The time distance between source event se of occurrence s and destination event de of occurrence d must be greater than or equal to δ . The notation used is $lbc(s, se, d, de, \delta)$.
- **Upper bound constraint:** The time distance between source event se of occurrence s and destination event de of occurrence d must be smaller than or equal to δ . The notation used is $ubc(s, se, d, de, \delta)$.

Explicit time constraints (temporal relationships and fixed-date bindings) can obviously only be reasonably defined in the model stratum (level) of a workflow, seeing that the model level of a workflow represents the expanded structure of its corresponding workflow specification. Given that and the fact that multiple models can be derived from a specification, explicit time constraints affect only a specific workflow model.

In contrast to [Bettini et al., 2002; Marjanovic and Orłowska, 1999b], our method allows to specify explicit constraints between parallel activity occurrences.

An example of a lower-bound constraint includes a legal workflow with activities of serving a warning and closing a business with the requirement that a certain time period passes between serving the warning and closing the business. Another example is a chemical process control workflow where a reaction is initiated only when certain time passes after the start of another reaction.

Upper-bound constraints are even more common. Thus, final patent filing must be done within a certain period of time after the preliminary filing. Like-

wise, time limits for responses to business letters provide typical examples of such constraints [Eder and Panagos, 2000].

We assume that for all upper and lower bound constraints the source node is *temporally* before the destination node according to the ordering implied by the workflow model.

6.4.2.2 Checking Constraint Correctness

Furthermore, temporal relationships are evidently restricted to occurrences *within* an instance type. However, our workflow modeling concept represents several instance types within a workflow model (cp. Section 5.3), hence the scope of temporal relationship constraints has to be checked. This checking procedure is shown in the following Algorithm 6.1.

Algorithm 6.1 Checking Constraint Correctness

```

1:
2: ↓ source,destination:ModelControlOccurrence; ↑ validConstraint:boolean
3:
4: validConstraint := false
5: source_pred_set := source.getAllPredecessor()
6: destination_pred_set := destination.getAllPredecessor()
7:
8: if source ∈ destination_pred_set then
9:   validConstraint := true
10: else
11:   common_pred_set := NULL
12:   for all common_pred ∈ source_pred_set and
       common_pred ∈ destination_pred_set do
13:     common_pred_set := common_pred_set ∪ {common_pred}
14:   end for
15:   for all common_pred ∈ common_pred_set do
16:     if common_pred.getSuccessor() ∩ common_pred_set = {} then
17:       if common_pred.ocIsKindOf(ModelControlOccurrence) and
           common_pred.cntrPosition = 'start' and
           common_pred.type <> 'cond' and
           common_pred.type <> 'alt' then
18:         validConstraint := true
19:       end if
20:     end if
21:   end for
22: end if

```

Algorithm 6.1 has two input parameters (source and destination node of the constraint) and a boolean output parameter that indicates the correctness of the considered constraint. If the source node is a predecessor of the destination node, then both nodes correspond to the same instance type, and therefore, the constraint is correct and the algorithm returns true (see line 8-9). Otherwise, we have to check the latest common predecessor that is a split node (see line 10-16). If this split node is not an or-split and not an alt-split (line 17), the constraint is correct because its source and destination node affects the same instance type. If the split node is an or-split or an alt-split, the constraint spans over two instance types and the algorithm returns false. Figure 6.2 shows some examples; *ubc2* and *ubc3* are correct constraints, whereas *ubc1* is incorrect.

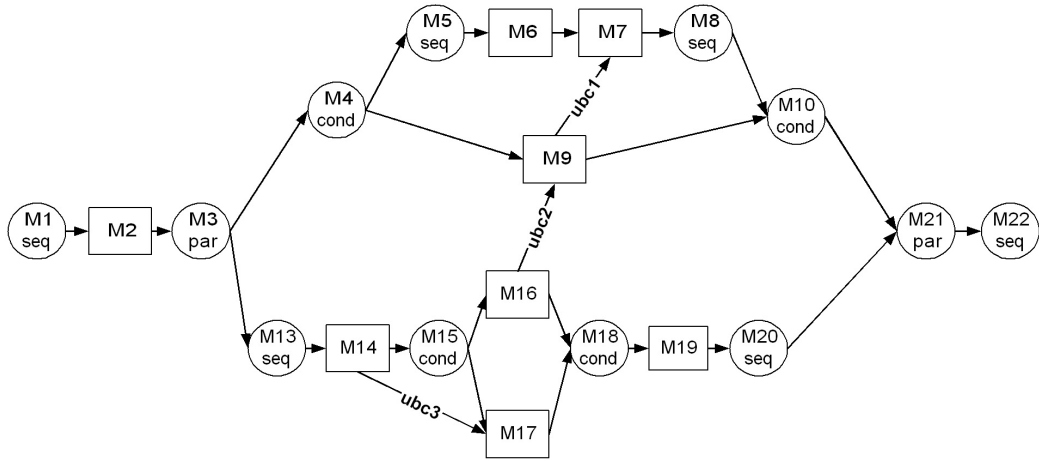


Figure 6.2: Examples of Correct and Incorrect Constraints

6.4.2.3 Fixed-Date Constraints

To express constraints that bind events to specific calendar dates, an abstraction that generalizes a, typically infinite, set of dates (*i.e.*, “every other Monday” or “every 5th workday of a month”) is required [Eder et al., 1999b; Eder and Panagos, 2000].

- **Fixed-date object:** A fixed-date object is an abstract data type T with the following methods: $T.next(D)$ and $T.prev(D)$ return, respectively, the next and previous valid dates after an arbitrary date D ; $T.period$ returns the maximum distance between valid dates; and $T.dist(T')$ returns the maximum distance between valid dates in the given object and in another fixed-date object T' , having as default value $T.period$.

Having fixed-date objects at our disposal, we can now define *fixed-date constraints* as follows [Eder et al., 1999b]:

- **Fixed-date constraint:** To express a time constraint that binds an event e of occurrence f to some fixed date(s), we write $fdc(f, e, T)$, where T is a *fixed-date object*.

Although several fixed-date constraints could be associated with an activity, for the sake of simplicity, we assume that only one such constraint is used in the remainder.

Conversion of Fixed-Date Constraints

An important aspect of the later computation is the transformation of fixed-date constraints into lower-bound constraints using worst-case estimates. This mapping is necessary because, at build time, calendar values for the workflow execution are not available and, thus, we can only use information about the duration between two valid time points for a fixed-date object. At process-instantiation time we will have more information concerning the actual delays due to fixed-date constraints [Eder and Panagos, 2000; Eder et al., 1999b].

Consider a fixed-date constraint $fdc(a, s, T)$. Assume that occurrences start instantaneously after all their predecessors finish. In the worst case, occurrence a may finish at $T.period + a.d$ after its last predecessor occurrence finishes, if the event s is the start. Let t_1 and t_2 be valid dates in T with the maximum time-interval between them. i.e., $t_2 - t_1 = T.period$, and let b be the last predecessor occurrence to finish (cp. [Eder and Panagos, 2000; Eder et al., 1999b]).

The time interval between end-events of b and a is the longest if b finishes just after time t_1 , because then a cannot start immediately (it would not finish at valid date $t_1 + a.d$), and would have to wait until time t_2 before starting. In this case, the distance between end events of b and a is $\delta = (t_2 - t_1) + a.d = T.period + a.d$, assuming b itself does not have a fixed-date constraint associated with it. If b has a fixed-date constraint $fdc(b, e, T')$, similar reasoning can be used to obtain $\delta = T.dist(T')$ if $a.d \leq T.dist(T')$ and $\delta = a.d + T.period$ otherwise (cp. [Eder and Panagos, 2000; Eder et al., 1999b]). The graphical representation of that issue is shown in Figure 6.3.

A fixed-date constraint $fdc(a, e, T)$ is replaced by lower-bound constraints $lbc(b, e, a, e, \delta)$ for every predecessor b of occurrence a , where δ is computed for each predecessor, as shown above.

Fixed-Date Constraints Associated with Start-Occurrences

If a fixed-date constraint is associated with the start occurrence, then a dummy occurrence has to be inserted as predecessor (cf. [Kolmann, 2001]), since the

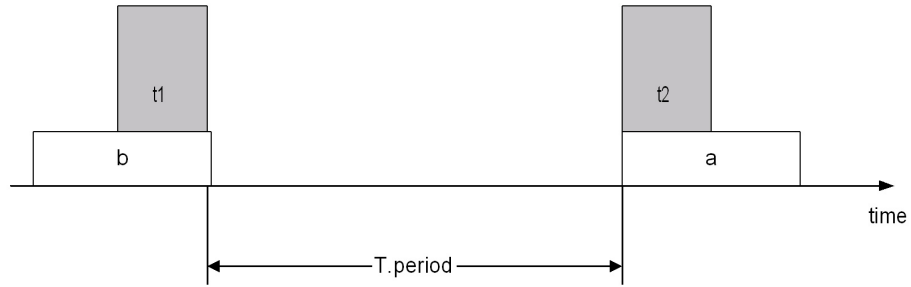


Figure 6.3: Conversion of Fixed-Date Constraints

derived lower bound constraint needs a relationship from a source node (here the dummy occurrence) to the destination node (here the start node).

6.5 Timed Workflow Model

Bearing in mind the time constraints introduced above, our time management techniques are based on the notion of a *timed workflow model*, which extends the workflow model of Section 3.3 by augmenting each activity A at the specification level with a deterministic discrete duration value and each occurrence n at the model level with the following.

- $n.E^{BF}$: The earliest point in time n can finish when the shortest conditional and alternative path is chosen to reach n ;
- $n.E^{BS}$: The earliest point in time n can finish when the shortest conditional and the longest alternative path is chosen to reach n ;
- $n.E^{WF}$: The earliest point in time n can finish when the longest path is chosen to reach n ;
- $n.E^{WS}$: The earliest point in time n can finish when the longest path is chosen to reach n ;
- $n.L^{BF}$: The latest point in time n has to finish in order to meet the overall deadline via the shortest conditional and alternative path;
- $n.L^{BS}$: The latest point in time n has to finish in order to meet the overall deadline via the shortest conditional and the longest alternative path;
- $n.L^{WF}$: The latest point in time n has to finish in order to meet the overall deadline via the longest conditional and shortest alternative path.
- $n.L^{WS}$: The latest point in time n has to finish in order to meet the overall deadline via the longest conditional and the longest alternative path.

For the sake of simplicity we restrict ourselves to present only time values for the end event of occurrences, since activity durations are assumed to be deterministic, and start times for occurrences are computed by subtracting their durations from their termination times.

The interpretation of this diagram is that E-values and L-values are the accumulation of possible execution paths. Only the best and worst, and the fastest and slowest execution paths are tracked explicitly, all others are included in the diagram. It is easy to see that the following invariants holds at any time for any occurrence s :

$$s.E^{BF} \leq s.E^{WF} \leq s.E^{WS}, s.E^{BF} \leq s.E^{BS} \leq s.E^{WS}, \text{ and } s.E^{WS} \leq s.L^{WS}$$

6.5.1 Timed Workflow Metamodel

Due to the temporal extensions of the workflow model, the metamodel has to be adapted to fit the changes. Hence, the metamodel of Section 4.4 is extended by the class *TimeConstraint* as well as by some additional attributes in the class *ModelElement*.

The association class *TimeConstraint* contains the necessary information for explicit time constraints. It has the unique attribute *tcId* for the identification, and the attribute *type* to identify lower and upper bound constraints. The attributes *bound* and *satisfied* complete the information. The involved occurrences (source und destination) are identified by the reflexive relationship of the association class referencing the class *ModelElement*.

Moreover, the class *ModelElement* is augmented with the attributes for the *E-values* and *L-values*. Figure 6.4 shows the extended metamodel.

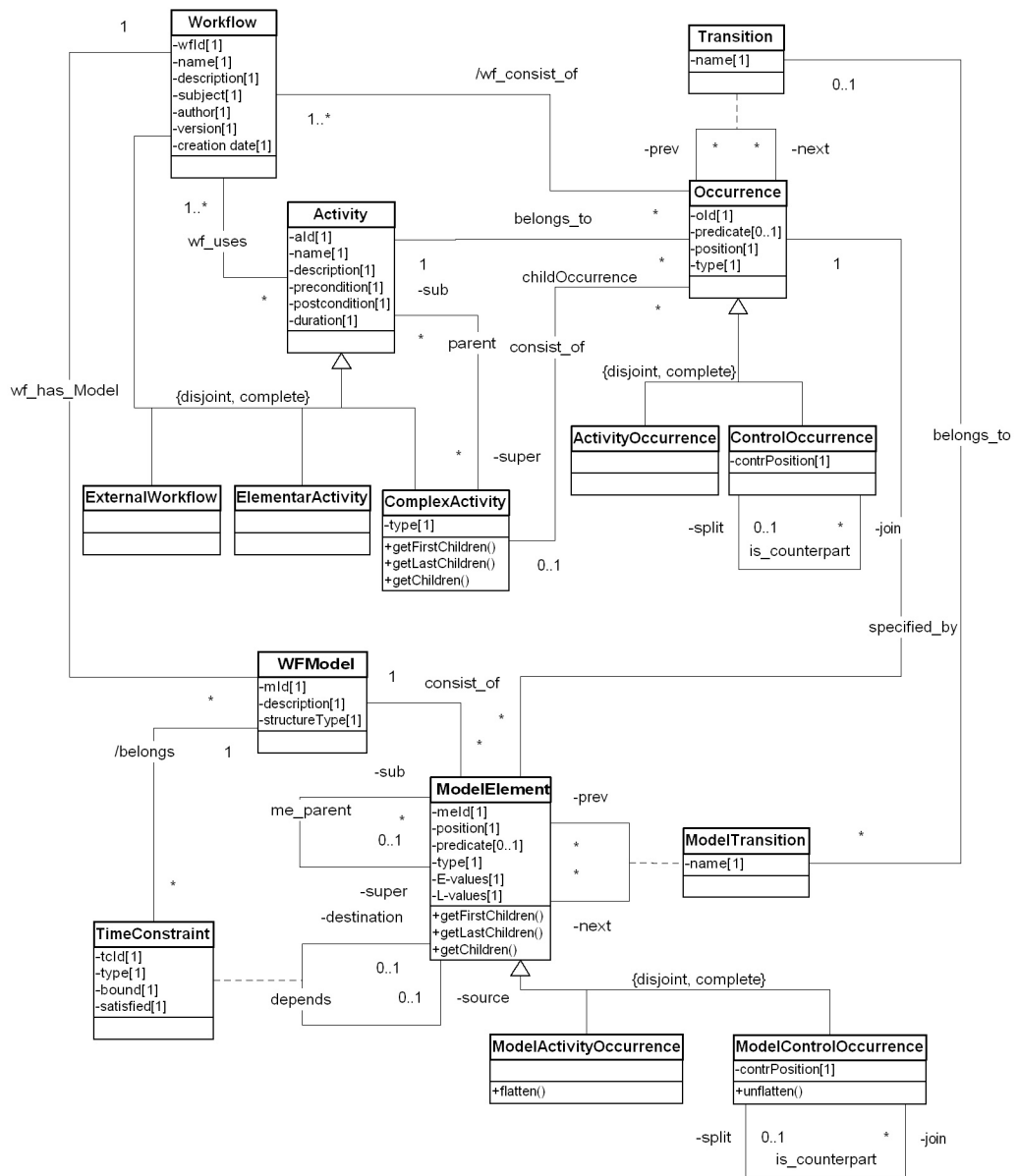


Figure 6.4: Timed Workflow Metamodel with Time Components

6.6 Representations of Timed Workflow Models

Before we move on to the calculation operations for the time values, we present some different representation methods for timed workflow models. Again, (timed) workflow models could be represented either in textual or in graph based forms (see Section 3.4). Process management already includes some common graph-based representations like PERT, CPM and Gantt Charts which are briefly introduced below.

There are many advantages to using net (graph-based) representation methods (cf. [Pozewaunig et al., 1997]):

- Net methods are widely used in project and workflow management.
- Time dependencies between parts of the workflow can be visualized immediately. Some types of restrictions can be identified by just looking at the diagram, and important aspects can easily be localized.
- Net diagrams are flexible and understandable. Adaption can easily be made, and they are traceable for every one.
- The accordance between workflow description languages (WDLs) and net methods facilitates the transformation from one concept to another without great effort. Additionally, it is also easy to interpret the values of the net diagram in terms of the workflow description.

6.6.1 PERT Chart

Program evaluation and review technique (PERT) charts depict task, duration, and dependency information. Each chart starts with an initiation node where the first task, or tasks, originates from. If multiple tasks begin at the same time, they are all started from the node or branch, or fork from the starting point. Each task is represented by a line which states its name or another identifier (known as *Activity-on-Arc (AoA)*²), its duration, the number of people assigned to it, and in some cases, the initials of the personnel assigned to it. The other end of the task line is terminated by another node which identifies the start of another task, or the beginning of any slack time, i.e. waiting time between tasks. Each task is connected to its successor tasks thereby forming a network of nodes and connecting lines. The chart is complete when all final tasks come together at the completion node. When slack time exists between the end of one task and the start of another, the usual method is to draw a broken or dotted line between the end of the first task and the start of the next dependent task [Modell, 1996]. A PERT chart may have multiple parallel or interconnecting networks of tasks, but no conditional

²A second representation type is the Activity-on-Node (AoN).

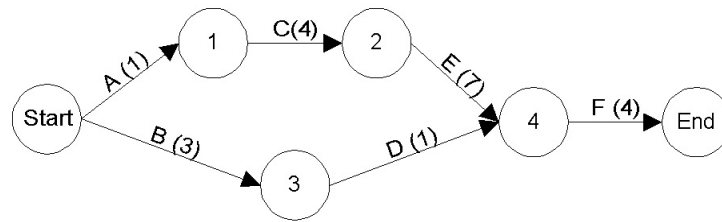


Figure 6.5: Example PERT Chart

execution of tasks. Figure 6.5 depicts such a task net where only the name and duration of each task is stated.

6.6.2 CPM Chart

Critical Path Method (CPM) charts are similar to PERT charts and are sometimes known as PERT/CPM. In a CPM chart, the critical path is indicated. A critical path consists of a set of dependent tasks (each one dependent on the preceding one) which together take the longest time to complete. Although it is unusual, a CPM chart can define multiple, equally critical paths. Tasks which fall on the critical path should be noted in some way, so that they attract special attention. One way to do this is to draw critical path tasks with a double line instead of a single line. Tasks which fall on the critical path should receive special attention by both the project manager and the personnel assigned to them [Modell, 1996].

6.6.3 Gantt Chart

A *Gantt chart* is a horizontal bar chart developed as a production control tool in 1917 by Henry L. Gantt, an American engineer and social scientist. Frequently used in project management, a Gantt chart provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific tasks in a project.

A Gantt chart is a matrix which on the vertical axis lists all the tasks to be performed. Each row contains a single task identification which usually consists of a number and a name. The horizontal axis is headed by columns indicating estimated task duration, skill level needed to perform the task, and the name of the person assigned to the task, followed by one column for each period in the project's duration. Each period may be expressed in hours, days, weeks, months, or other time units. In some cases it may be necessary to label the period columns as period 1, period 2, and so on. The graphics portion of the Gantt chart consists of a horizontal bar for each task connecting the period start and period ending columns. A set of markers is usually used to indicate estimated and actual start and end. Both each bar and the name of each person assigned to the task are on a separate line [Modell, 1996]. Figure 6.6 shows an example for a Gantt Chart.

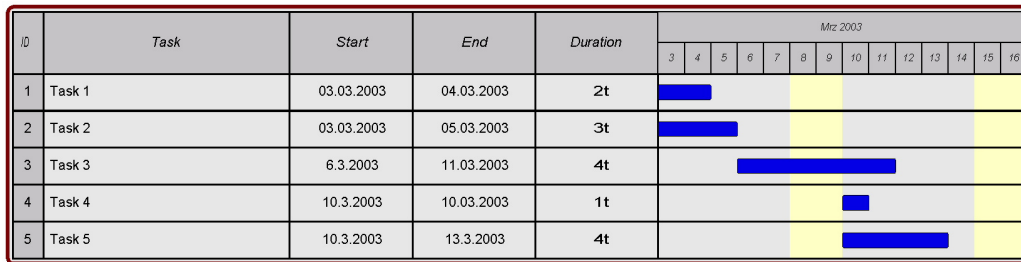


Figure 6.6: Gantt Chart Example

6.6.4 ePERT Chart

Several concepts needed to represent workflows are not covered by the “classical” PERT-diagram. One of the most important of these concepts is the notion of alternative execution paths a process can take. This cannot be modeled by means of a net diagram. In fact, the only way is to design a separate plan for each possible execution path. Because of the complexity of an average workflow structure and the exploding number of separate net diagrams, this is in general not acceptable. Moreover, it is necessary to allow the transformation of further control structures, like iteration or choices. That is why a new syntactical element needs to be introduced directly to the syntax of PERT, the alternative. Net diagrams with alternatives are called *extended PERT* or ePERT. Alternative paths in ePERT are indicated by an arc which spans over the edges of the succeeding activities. In an ePERT Chart, every node stores four time values for the earliest and latest finish time [Pozewaunig et al., 1997; Pozewaunig, 1996]. Figure 6.7 shows an example for an ePERT Chart.

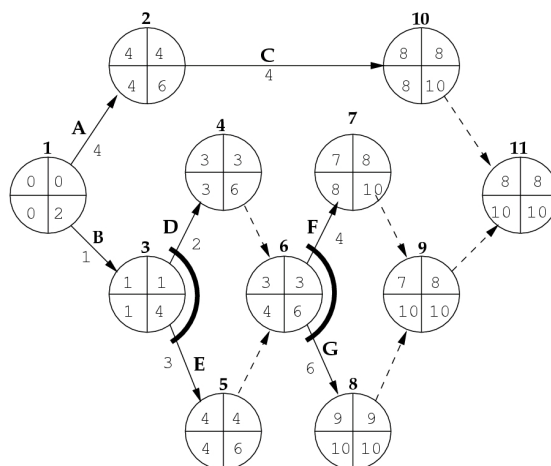


Figure 6.7: Example ePERT Chart acc. to [Pozewaunig et al., 1997]

6.7 Timed Workflow Graph

The main goal of a workflow graph is to show the control and data flow between the activities of a given workflow process. Workflow graphs (see Section 3.5.2) can be extended to include time-related data. Since workflow graph nodes, which correspond to activity occurrences, have attributes associated with them such as the role that is responsible for the enactment of the activity, one could easily model time-related data (see Section 6.5) by adding more attributes. Moreover, time constraints between activity occurrences or control occurrences could be shown in the graph by additional edges, such as *time edges* [Dadam et al., 2000].

In the following, we focus on time-related activity occurrence attributes. A workflow graph that includes time information is referred to as *timed activity graph* or *timed workflow graph (TWfG)*. Each activity occurrence node in a timed graph is called *timed activity occurrence node* [Eder and Panagos, 2000].

As we have mentioned in the previous Section, each activity A has *start* and *end events* associated with. Depending on the execution duration(s) associated with A , several pairs of these events could be “attached” to A . For the sake of simplicity, however, we use the average execution time as the expected execution duration of an activity.

Duration	
E^{BF}	L^{BF}
E^{BS}	L^{BS}
E^{WF}	L^{WF}
E^{WS}	L^{WS}

Figure 6.8: An Occurrence Node in a Timed Workflow Graph

Here, the start event can be computed when the end event is known and, thus, we only need to consider end events when modeling time constraints (cp. [Eder et al., 2000 1999a]). According to this, each activity occurrence is augmented with eight time values as presented in Section 6.5. Figure 6.8 shows the time extension of such a node.

6.7.1 Constraint Representation

None of the representation methods of Section 6.6 can cope with explicit time constraints. Therefore, we developed an appropriate time constraint representation.

For a temporal relationship constraint a labelled directed edge with a dot and dash line that we call *time constraint edge (TCE)* is placed in the TWfG. The

time constraint edge originates from the source node to the destination node of the belonging constraint, and it is labelled with the constraint id, constraint type, bound and the satisfied attribute. Time constraint edges are similar to time edges in [Dadam et al., 2000].

Figure 6.9 shows an example timed workflow graph that is based on the aforementioned notions. Our method for the computation of activity occurrence end events is presented in the next Section.

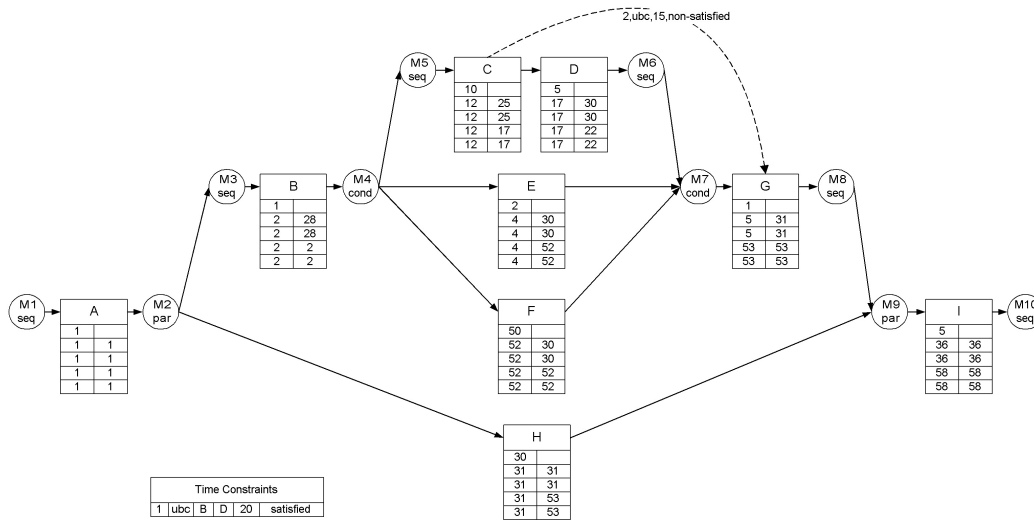


Figure 6.9: Example Timed Workflow Graph

An alternative way to depict explicit time constraints in timed workflow graphs is, for the sake of clarity, to tabulate. Such a table is like a legend. Both representation methods can be mixed freely. For better comprehensibility it is convenient to represent unsatisfied time constraints in graphical form and the satisfied ones in tabular form.

6.8 Timed Workflow Model Calculus

From the area of operations research [Glover et al., 1992; Philipose, 1986; Eiselt and von Frajer, 1977] or production informatics [Galloway, 1993; Dorninger et al., 1990] several techniques are known to calculate and optimize parameters of process resources. By analyzing these techniques we found several similarities between workflow modeling and execution on the one side and project planning methods on the other. In our work we use a method of net optimization, PERT (Program Evaluation and Review Technique) or ePERT, to take the information obtained by the workflow definition and calculate buffer times for every activity occurrence [Pozewaunig et al., 1997].

For the sake of simplicity, we assume that all time information is given in some basic time units. For applications, time information has to be given in application specific temporal units. For build time and workflow schemes, time information is always given relative to the beginning of a workflow. For workflow instances, this time information is mapped to an actual calendar.

In this and the following Sections, we bring together the timed workflow graph calculations of [Eder et al., 1999a; Pozewaunig et al., 1997], which are based on PERT/CPM, and the time constraint incorporation method of [Eder et al., 1999b] that is based on CPM. That has already been done in [Eder et al., 2000] with a different metamodel as introduced in Section 4.4. Hence, an adaption of the calculation operations as mentioned above is performed. Subsequently, some considerations about these presumed calculation techniques are made briefly.

Without explicit time constraints, E- and L-values can be computed by extending the Critical Path Method (CPM) [Philipose, 1986] to handle conditional [Pozewaunig et al., 1997] and alternative execution paths [Eder et al., 1999a].

CPM assumes that the durations of activities are deterministic. We are aware that this assumption does not hold true for many workflows and that for these workflows a technique dealing with a probability distribution of activity durations like the Project Evaluation and Review Technique (PERT) [Philipose, 1986] would be more appropriate. However, we chose the CPM technique as it allows us to present the concept more clearly without the math involved with probability distributions.

All calculations are accomplished only on full flattened models because complex activity occurrences may have different execution durations due to conditionals or alternatives. That issue is dealt with in [Eder and Pichler, 2002] by means of *duration histograms*.

6.8.1 Computation of the Timed Workflow Model

We first compute the timed graph using structural constraints. Table 6.1 shows the actual computations, where $d(n)$ denotes the execution duration of occurrence n . E-values are computed in a forward pass, with the E-values of the starting workflow occurrence being set to its duration. L-values are computed in a backward pass, with the L-values of the last workflow occurrence equal to its E-values or to its max E-value.

The distance between the E-value and the L-value of an activity occurrence is its *buffer time* or *slack time*. In our example, occurrence D has a buffer of 35 time units in all cases ($BF, BS, WF, and WS$). This buffer, however, is not exclusively available to only one occurrence. In our example, the buffer of D is shared with C . If C uses some buffer-time, then the buffer of D is reduced.

Best Case		
forward	(best fastest)	(best slowest)
<i>sequence</i>	$j.E^{BF} = \tau.E^{BF} + d(j)$	$j.E^{BS} = \tau.E^{BS} + d(j)$
<i>and-join</i>	$j.E^{BF} = \max(\{\tau.E^{BF} + d(j)\})$	$j.E^{BS} = \max(\{\tau.E^{BS} + d(j)\})$
<i>or-join</i>	$j.E^{BF} = \min(\{\tau.E^{BF} + d(j)\})$	$j.E^{BS} = \min(\{\tau.E^{BS} + d(j)\})$
<i>alt-join</i>	$j.E^{BF} = \min(\{\tau.E^{BF} + d(j)\})$	$j.E^{BS} = \max(\{\tau.E^{BS} + d(j)\})$
	\forall immediate activity occurrence predecessor τ of j	
reverse	(bf)	(bs)
<i>sequence</i>	$j.L^{BF} = \tau.E^{BF} - d(\tau)$	$j.L^{BS} = \tau.E^{BS} - d(\tau)$
<i>and-split</i>	$j.L^{BF} = \min(\{\tau.E^{BF} - d(\tau)\})$	$j.L^{BS} = \min(\{\tau.E^{BS} - d(\tau)\})$
<i>or-split</i>	$j.L^{BF} = \max(\{\tau.E^{BF} - d(\tau)\})$	$j.L^{BS} = \max(\{\tau.E^{BS} - d(\tau)\})$
<i>alt-split</i>	$j.L^{BF} = \max(\{\tau.E^{BF} - d(\tau)\})$	$j.L^{BS} = \min(\{\tau.E^{BS} - d(\tau)\})$
	\forall immediate activity occurrence successor τ of j	
Worst Case		
forward	(wf)	(ws)
<i>sequence</i>	$j.E^{WF} = \tau.E^{WF} + d(j)$	$j.E^{WS} = \tau.E^{WS} + d(j)$
<i>and-join</i>	$j.E^{WF} = \max(\{\tau.E^{WF} + d(j)\})$	$j.E^{WS} = \max(\{\tau.E^{WS} + d(j)\})$
<i>or-join</i>	$j.E^{WF} = \max(\{\tau.E^{WF} + d(j)\})$	$j.E^{WS} = \max(\{\tau.E^{WS} + d(j)\})$
<i>alt-join</i>	$j.E^{WF} = \min(\{\tau.E^{WF} + d(j)\})$	$j.E^{WS} = \max(\{\tau.E^{WS} + d(j)\})$
	\forall immediate activity occurrence predecessor τ of j	
reverse	(wf)	(ws)
<i>sequence</i>	$j.L^{WF} = \tau.E^{WF} - d(\tau)$	$j.L^{WS} = \tau.E^{WS} - d(\tau)$
<i>and-split</i>	$j.L^{WF} = \min(\{\tau.E^{WF} - d(\tau)\})$	$j.L^{WS} = \min(\{\tau.E^{WS} - d(\tau)\})$
<i>or-split</i>	$j.L^{WF} = \min(\{\tau.E^{WF} - d(\tau)\})$	$j.L^{WS} = \min(\{\tau.E^{WS} - d(\tau)\})$
<i>alt-split</i>	$j.L^{WF} = \max(\{\tau.E^{WF} - d(\tau)\})$	$j.L^{WS} = \min(\{\tau.E^{WS} - d(\tau)\})$
	\forall immediate activity occurrence successor τ of j	

Table 6.1: Calculation instructions for timed workflow model without explicit time constraints

Computing the timed workflow graph delivers the duration of the entire workflow, and deadlines for all occurrences so that the termination of the entire workflow is not delayed. Thus the timed activity graph gives the necessary information for the dispatching of occurrences at run time.

Figure 6.10 shows the timed workflow graph we use in the rest of the thesis. In this graph, activity *A* is followed by an and-split, having *M9* as the and-join, and activity *B* is followed by an or-split, having *M7* as the or-join. The values of node *I* show the duration of the workflow, i.e. $I.E^{BF}$ and $I.E^{WS}$ indicate that the workflow execution may take between 36 and 58 time units. $G.E^{BF}$ indicates that *G* may finish after 5 time units (when *E* follows *B*), while $G.E^{WS}$ indicates that no path from *A* to *G* should take more than 53 time units. $B.L^{BF}$, tells us that if *B* is finished at time point 50, the workflow may still be able to terminate in time. However, we do not have any guaranty, since we do not know which conditional paths will be followed afterwards. From $B.L^{WS}$ we learn that if *B* is finished at time point 2, we can meet the overall deadline, irrespective of the conditionals.

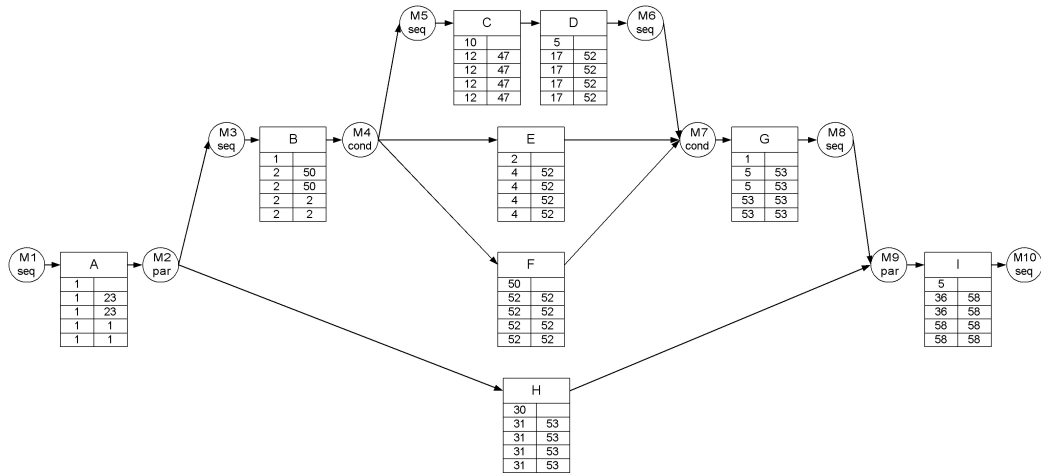


Figure 6.10: Example timed workflow graph without upper bound constraints

6.9 Time Constraint Satisfiability

After activity occurrence durations and time constraints are assigned, time calculations are required for computing optimistic and pessimistic start and finish times of activity occurrences within processes, available slack time for activities, updating existing deadlines, converting relative time information to absolute time points, and so on. Usually the assignment of external deadlines (explicit time constraints) is an iterative process. The designer first assigns the duration to activities,

and the time calculations at process build time are then used to compute the duration of the whole process and the relative position of all activity occurrences. The designer can then choose to set external deadlines to some of the activity occurrences and recompute the time information. If external deadlines cannot be met, the designer might modify the workflow structure or change the deadlines [Eder and Panagos, 2000].

We suggest that a timed workflow graph satisfies a time constraint if the executions in which all activities complete at their E-values or L-values are valid with respect to this constraint. Here, we demand for equal treatment of conditionals and alternatives in the context of time constraints. For explicit time constraints this means (cf. [Eder et al., 2000]):

- *upper-bound constraints* $ubc(s, d, \delta)$ should check for constraint violations; for $s.E^X + \delta \leq d.E^X$ and $s.L^X + \delta \leq d.L^X$, where $X \in \{BF, BS, WF, WS\}$.
- *lower-bound constraints* $lbc(s, d, \delta)$ should check for constraint violations; for $s.E^X + \delta \geq d.E^X$ and $s.L^X + \delta \geq d.L^X$, where $X \in \{BF, BS, WF, WS\}$.

When a constraint is violated, the E- and L-values of source- and destination occurrence are shifted in an attempt to satisfy the constraint, with the invariant that an E-value is not greater than its corresponding L-value [Eder et al., 2000].

6.10 Incorporating Explicit Time Constraints

In this Section, we outline a technique that can be used to verify time constraint satisfiability, *i.e.*, it is possible to find a workflow execution that satisfies all time constraints.

Once the timed workflow graph is constructed, we can incorporate explicit time constraints into it by using the algorithms shown in [Eder et al., 2000 1999b]. Lower bound constraints are incorporated during the construction of the timed workflow graph; they may increase E-values during the forward pass and decrease L-values during the reverse pass. Additionally, the incorporation of upper-bound constraints should check for constraint violations (see above Section). When an upper bound constraint is violated, the E-values and L-values of source and destination occurrence are shifted in an attempt to satisfy the constraint with the invariant that an E-value is not greater than its corresponding L-value [Eder et al., 2000].

During the incorporation of explicit constraints the semantics of the E-values change: the E-values mandate that activity terminations should not occur before them in order to meet the time constraints. Therefore, the E-values and L-values define the time interval during which an activity has to terminate. This time interval is referred to as the *life-line* of the activity [Eder et al., 2000].

However, [Eder et al., 1999b] do not distinguish between conditional and unconditional (parallel) branches in the computation of worst case E- and L-values. While this ensures that the execution of the workflow will avoid to violate temporal constraints when the incorporation algorithm succeeds, it is too pessimistic. In particular, there are cases where execution without constraint violation is possible and the incorporation algorithm does not succeed due to the interference of constraints on mutually exclusive conditional branches, as we will show below [Eder et al., 2000].

6.10.1 Conditional Execution Paths

When explicit time constraints involve conditionally executed occurrences, it may be beneficial to consider some/all of the conditional paths in isolation. By doing so, we may be able to avoid superfluous constraint violations and to schedule conflicts during process execution. In general, the following issues need to be addressed when we derive timed graphs that violate explicit time constraints [Eder and Panagos, 2000; Eder et al., 2000]:

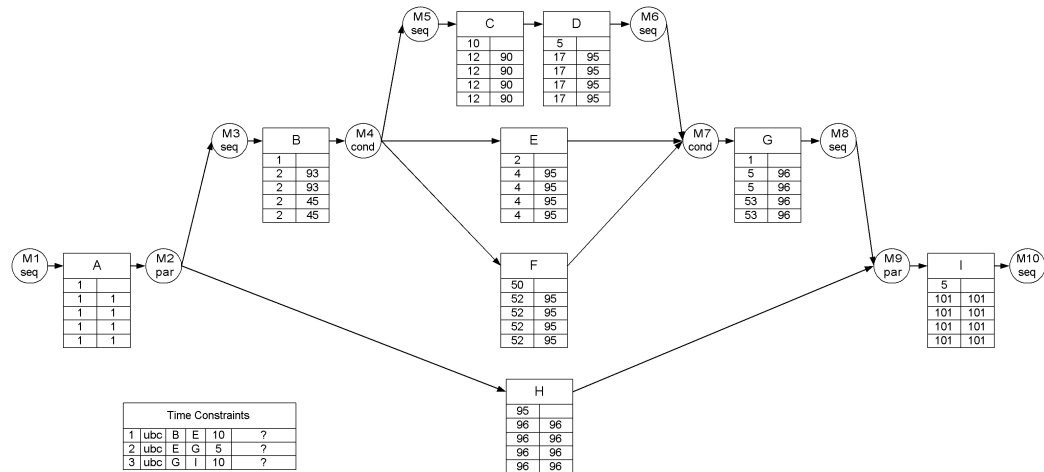


Figure 6.11: Workflow Example with Conditional Execution Paths

- 1 *Checking individual constraints for violation may not be sufficient.* As shown in [Eder et al., 1999b], a set of time constraints may not be satisfiable, even if each individual constraint is satisfiable. Consequently, the incorporation procedure should consider all of the constraints.
- 2 *Checking workflow instance types for constraint violation in isolation is not sufficient.* If two instance types only differ after or-splits, the common initial

activities should have the same E- and L-values. If we cannot find such E- and L-values in all instance types to satisfy the constraints, then it may not be possible to schedule the execution of this workflow in a way that all time constraints are met.

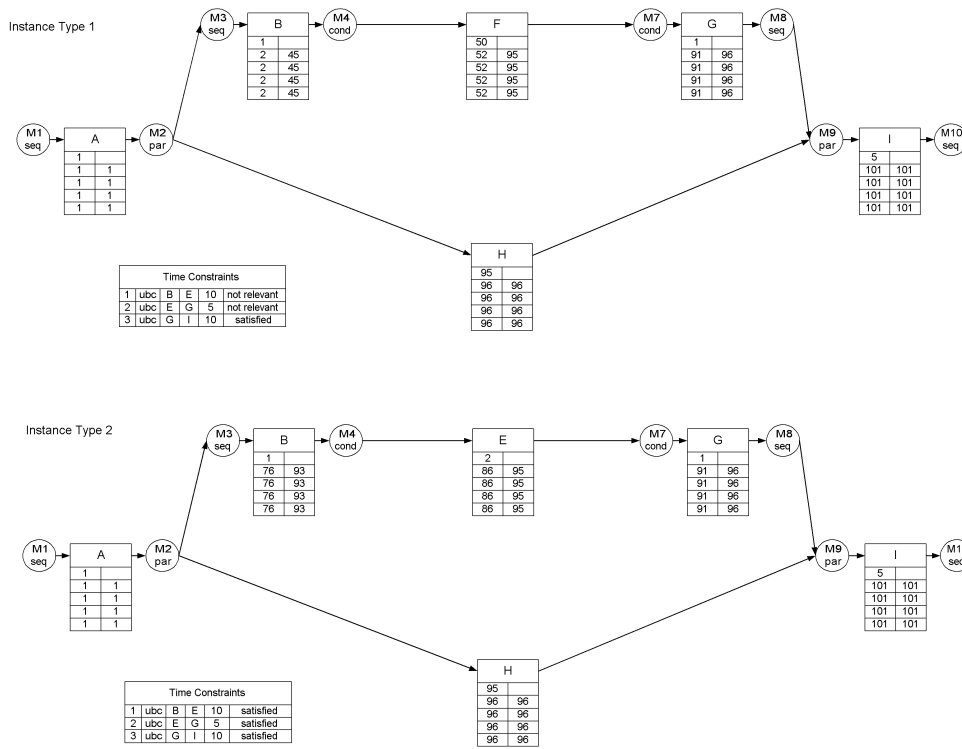


Figure 6.12: Workflow Instance Types of Figure 6.11

If we consider two instance types as shown in Figure 6.12 for the workflow in Figure 6.11 (duration of occurrence H is changed to 95 as well as modified time constraints), where one includes F and results in $B.E^X = 2$ and $B.L^X = 45$, and the other one includes E and results in $B.E^X = 76$ and $B.L^X = 93$ ($X \in \{BF, BS, WF, WS\}$). Here, it is not possible to satisfy the time constraints for either instance, since B is executed in both of them, and the scheduling information for B , i.e., the valid interval for B to terminate, is contradictory. Consequently, we cannot start executing B since we do not know which path is going to be taken after it.

- 3 *Incorporating upper-bound constraints using best-case values may not be meaningful.* When an upper bound constraint exists between a conditionally executed activity C and a successor activity G , which is always executed, check-

ing this constraint for the best-case is not possible when the E- and L-values of G do not depend on the best-case E- and L-values of C .

- 4 *Checking violation of upper-bound constraints using worst-case values may lead to unnecessary rejections when the workflow has conditional branches.* Similar to the above case, when the worst-case E- and L-values of C do not contribute to the worst-case values of another activity, we may find a constraint violation when trying to incorporate several constraints.

This can be seen when examining the case where $ubc(B, D, 20)$ and $ubc(C, G, 15)$ need to be incorporated into the timed graph shown in Figure 6.9. If we incorporate $ubc(B, D, 20)$ first, then $D.L^{wc}$ becomes 22 and, consequently, $ubc(C, G, 15)$ cannot be satisfied. However, each of these constraints is individually satisfiable, as shown in Figure 6.13, since the path containing C and D does not influence the computation of the worst case E- and L-values of nodes B and G .

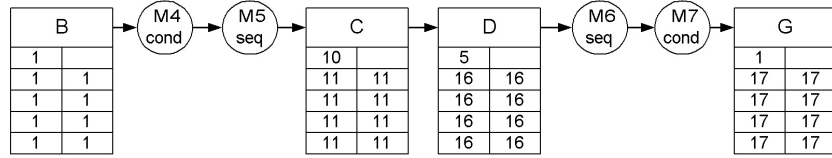


Figure 6.13: Path with $ubc(B, D, 20)$ and $ubc(C, G, 15)$

6.10.2 Calculation/Incorporation Procedure

Initially, two passes over the full flattened workflow graph are performed. We first compute the timed graph using structural constraints and any explicit lower-bound constraints. The E- and L-values for all activity occurrences are computed using an extension of the CPM method. In particular, E-values of occurrences without predecessors are set to the durations of these activities, and a forward traversal of the workflow graph is done for computing the remaining E-values. Then, the L-values of activities without successors are set to their corresponding E-values, and a backward traversal of the workflow graph is done for computing the remaining L-values. During this traversal, if external lower bound constraints exists, the E- and L-values of the activity occurrences are correspondingly modified as can be seen in Section 6.10.3.

Then, we attempt to incorporate all upper-bound constraints. Upper-bound constraints are incorporated into the computed E- and L-values. A necessary condition for the constraint $ubc(s, d, \delta)$ to be satisfiable is that the *distance* between s and d is less than δ . This distance is the sum of the execution durations of the occurrences on the longest path between s and d . Since this distance only depends

on the E- and L-values of s and d , a violated upper-bound constraint could be satisfied by changing these values in a consistent way. Details about how such changes are performed as well as the algorithmic properties of our technique can be found in Section 6.10.4. It should be noted, though, that the satisfaction of individual upper-bound constraints may lead to a violation of already incorporated upper-bound constraints. Therefore, when an upper-bound constraint is incorporated into the E- and L-values of activities, all previously incorporated upper-bound constraints should be validated again.

When an upper-bound constraint is violated, we determine whether its source and destination nodes are connected via conditionally executed activities or whether they belong to the same workflow instance type (see Algorithm 6.1). Here, we partially unfold the workflow graph and, finally, we attempt the constraint incorporation procedure again (cp. [Eder and Panagos, 2000; Eder et al., 2000]).

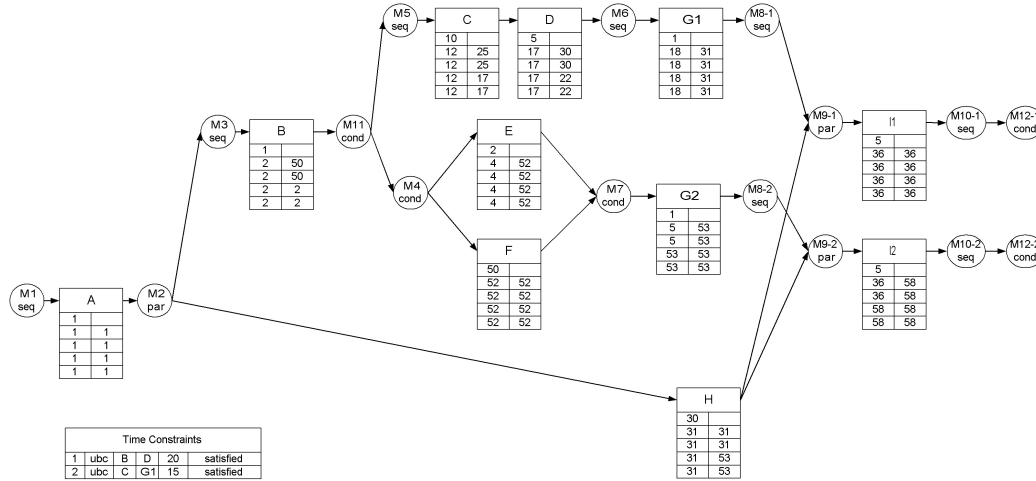


Figure 6.14: Partially Unfolded Workflow of Figure 6.9

To check the satisfiability of time constraints, we extended the algorithm in [Eder et al., 2000 1999b] to the full nodes (best/worst and fastest/slowest cases). If a constraint $ubc(s, d, \delta)$ cannot be successfully incorporated, the algorithm partially unfolds the workflow starting from s and d , if unfolding has not yet happened. After unfolding, we restart the incorporation procedure on the partially unfolded workflow graph. That procedure is shown with Algorithm 6.4.

If a constraint is violated despite unfolding, we check whether there is an overlapping constraint and perform the unfold for the source and destination nodes of this constraint. An example for this procedure is given in the workflow shown in Figure 6.9 and the constraints $ubc(B, D, 20)$ and $ubc(C, G, 15)$. If $ubc(B, D, 20)$

is incorporated first, then $ubc(C, G, 15)$ cannot be incorporated since $D.L^{WS}$ is 22 (and therefore $C.L^{WS}$ is 17). Now we partially unfold the workflow graph, compute the timed graph, and incorporate the constraints there. The result of this procedure is the graph shown in Figure 6.14.

As mentioned above, the following calculation algorithms are defined on full flattened workflow graphs.

6.10.3 Lower-Bound Constraint

Algorithm 6.2 shows computation (forward and backward) of the timed graph using structural constraints and any explicit lower-bound constraints. In the actual computations $n.d$ denotes the execution duration of occurrence n .

Forward Calculations

```

for all occurrences a with a.pos = start
  a.EBF := a.d
  a.EBS := a.d
  a.EWF := a.d
  a.EWS := a.d
endfor
for all occurrences a with a.pos ≠ start
  in a topological order
  switch a.type
  case 'seq':
    a.EBF := max(b.EBF + a.d,
                  {m.s.EBF + m.δ | m = lbc(s, a, δ)})
    a.EBS := max(b.EBS + a.d,
                  {m.s.EBS + m.δ | m = lbc(s, a, δ)})
    a.EWF := max(b.EWF + a.d,
                  {m.s.EWF + m.δ | m = lbc(s, a, δ)})
    a.EWS := max(b.EWS + a.d,
                  {m.s.EWS + m.δ | m = lbc(s, a, δ)})
  case 'and-join':
    a.EBF := max({b.EBF + a.d | b ∈ a.pred},
                  {m.s.EBF + m.δ | m = lbc(s, a, δ)})
    a.EBS := max({b.EBS + a.d | b ∈ a.pred},
                  {m.s.EBS + m.δ | m = lbc(s, a, δ)})
    a.EWF := max({b.EWF + a.d | b ∈ a.pred},
                  {m.s.EWF + m.δ | m = lbc(s, a, δ)})
    a.EWS := max({b.EWS + a.d | b ∈ a.pred},
                  {m.s.EWS + m.δ | m = lbc(s, a, δ)})
  case 'or-join':
    a.EBF := min({b.EBF + a.d | b ∈ a.pred},
                  {m.s.EBF + m.δ | m = lbc(s, a, δ)})
    a.EBS := min({b.EBS + a.d | b ∈ a.pred},

```

```

        {m.s.EBS + m.δ | m = lbc(s,a,δ)})
    a.EWF := max({b.EWF + a.d | b ∈ a.pred},
        {m.s.EWF + m.δ | m = lbc(s,a,δ)})
    a.EWS := max({b.EWS + a.d | b ∈ a.pred},
        {m.s.EWS + m.δ | m = lbc(s,a,δ)})
  case 'alt-join':
    a.EBF := min({b.EBF + a.d | b ∈ a.pred},
        {m.s.EBF + m.δ | m = lbc(s,a,δ)})
    a.EBS := max({b.EBS + a.d | b ∈ a.pred},
        {m.s.EBS + m.δ | m = lbc(s,a,δ)})
    a.EWF := min({b.EWF + a.d | b ∈ a.pred},
        {m.s.EWF + m.δ | m = lbc(s,a,δ)})
    a.EWS := max({b.EWS + a.d | b ∈ a.pred},
        {m.s.EWS + m.δ | m = lbc(s,a,δ)})
  otherwise:
    // split nodes, activity occurrences
    a.EBF := max(b.EBF + a.d,
        {m.s.EBF + m.δ | m = lbc(s,a,δ)})
    a.EBS := max(b.EBS + a.d,
        {m.s.EBS + m.δ | m = lbc(s,a,δ)})
    a.EWF := max(b.EWF + a.d,
        {m.s.EWF + m.δ | m = lbc(s,a,δ)})
    a.EWS := max(b.EWS + a.d,
        {m.s.EWS + m.δ | m = lbc(s,a,δ)})
  endswitch
endfor

```

Algorithm 6.2: Forward Calculation

Backward Calculations

```

for all occurrences a with a.pos = end
  a.LBF := a.EBF
  a.LBS := a.EBS
  a.LWF := a.EWF
  a.LWS := a.EWS
endfor
for all occurrences a with a.pos ≠ end
  in a reverse topological order
  switch a.type
  case 'seq':
    // similar calculation as and-split because of
    // possible multiple successors
    a.LBF := max({s.LBF - s.d | s ∈ a.succ},
                  {m.d.LBF - m.δ | m = lbc(a,d,δ)})
    a.LBS := max({s.LBS - s.d | s ∈ a.succ},
                  {m.d.LBS - m.δ | m = lbc(a,d,δ)})
    a.LWF := max({s.LWF - s.d | s ∈ a.succ},
                  {m.d.LWF - m.δ | m = lbc(a,d,δ)})
    a.LWS := max({s.LWS - s.d | s ∈ a.succ},
                  {m.d.LWS - m.δ | m = lbc(a,d,δ)})
  case 'and-split':
    a.LBF := min({s.LBF - s.d | s ∈ a.succ},
                  {m.d.LBF - m.δ | m = lbc(a,d,δ)})
    a.LBS := min({s.LBS - s.d | s ∈ a.succ},
                  {m.d.LBS - m.δ | m = lbc(a,d,δ)})
    a.LWF := min({s.LWF - s.d | s ∈ a.succ},
                  {m.d.LWF - m.δ | m = lbc(a,d,δ)})
    a.LWS := min({s.LWS - s.d | s ∈ a.succ},
                  {m.d.LWS - m.δ | m = lbc(a,d,δ)})
  case 'or-split':
    a.LBF := max({s.LBF - s.d | s ∈ a.succ},
                  {m.d.LBF - m.δ | m = lbc(a,d,δ)})
    a.LBS := max({s.LBS - s.d | s ∈ a.succ},
                  {m.d.LBS - m.δ | m = lbc(a,d,δ)})
    a.LWF := min({s.LWF - s.d | s ∈ a.succ},
                  {m.d.LWF - m.δ | m = lbc(a,d,δ)})
    a.LWS := min({s.LWS - s.d | s ∈ a.succ},
                  {m.d.LWS - m.δ | m = lbc(a,d,δ)})
  case 'alt-split':
    a.LBF := max({s.LBF - s.d | s ∈ a.succ},
                  {m.d.LBF - m.δ | m = lbc(a,d,δ)})
    a.LBS := min({s.LBS - s.d | s ∈ a.succ},
                  {m.d.LBS - m.δ | m = lbc(a,d,δ)})
    a.XLWF := max({s.LWF - s.d | s ∈ a.succ},
                  {m.d.LWF - m.δ | m = lbc(a,d,δ)})

```

```

    a.LWS := min({s.LWS - s.d | s ∈ a.succ},
                  {m.d.LWS - m.δ | m = lbc(a,d,δ)})
otherwise:
    // join nodes, activity occurrences
    a.LBF := min({s.LBF - s.d | s ∈ a.succ},
                  {m.d.LBF - m.δ | m = lbc(a,d,δ)})
    a.LBS := min({s.LBS - s.d | s ∈ a.succ},
                  {m.d.LBS - m.δ | m = lbc(a,d,δ)})
    a.LWF := min({s.LWF - s.d | s ∈ a.succ},
                  {m.d.LWF - m.δ | m = lbc(a,d,δ)})
    a.LWS := min({s.LWS - s.d | s ∈ a.succ},
                  {m.d.LWS - m.δ | m = lbc(a,d,δ)})
endswitch
endfor

```

Algorithm 6.3: Backward Calculation

6.10.4 Upper-Bound Constraints

Algorithm 6.4 shows the incorporation of all upper-bound constraints based on the timed graph as it results from Algorithms 6.2 and 6.3.

When we check slack time at m.s, we use the \geq relation. When we check slack time at m.d, we use the \leq relation, which is different to [Eder et al., 1999b] who use the relations $>$ and $<$.

```

repeat
    error := false
    for each m = ubc(s,d,δ)
        if (m.s.EBF + m.δ < m.d.EBF or m.s.EBS + m.δ < m.d.EBS or
            m.s.EWF + m.δ < m.d.EWF or m.s.EWS + m.δ < m.d.EWS)
            (* violation at E *)
            if (m.s.LBF ≥ m.d.EBF - m.δ and m.s.LBS ≥ m.d.EBS - m.δ and
                m.s.LWF ≥ m.d.EWF - m.δ and m.s.LWS ≥ m.d.EWS - m.δ)
                (* slack at m.s *)
                m.s.EBF := max(m.d.EBF - m.δ, m.s.EBF)
                m.s.EBS := max(m.d.EBS - m.δ, m.s.EBS)
                m.s.EWF := max(m.d.EWF - m.δ, m.s.EWF)
                m.s.EWS := max(m.d.EWS - m.δ, m.s.EWS)
                recompute timed graph
                if m.d.EBF or m.d.EBS or
                    m.d.EWF or m.d.EWS changes
                    graph_changed := unfold(s,d)

```

```

        if not graph_changed
            error := true      (* already unfolded *)
        else
            recompute timed graph
        endif
    endif
else
    error := true (* no slack *)
endif
endif
if (m.s.LBF + m.δ < m.d.LBF or m.s.LBS + m.δ < m.d.LBS or
    m.s.LWF + m.δ < m.d.LWF or m.s.LWS + m.δ < m.d.LWS)
    (* violation at L *)
    if (m.d.EBF ≤ m.s.LBF + m.δ and m.d.EBS ≤ m.s.LBS + m.δ and
        m.d.EWF ≤ m.s.LWF + m.δ and m.d.EWS ≤ m.s.LWS + m.δ)
        (* slack at m.d *)
        m.d.LBF := min(m.s.LBF + m.δ, m.d.LBF)
        m.d.LBS := min(m.s.LBS + m.δ, m.d.LBS)
        m.d.LWF := min(m.s.LWF + m.δ, m.d.LWF)
        m.d.LWS := min(m.s.LWS + m.δ, m.d.LWS)
        recompute timed graph
        if m.s.LBF or m.s.LBS or
            m.s.LWF or m.s.LWS changes
            graph_changed := unfold(s,d)
            if not graph_changed
                error := true      (* already unfolded *)
            else
                recompute timed graph
            endif
        endif
    else
        error := true      (* no slack *)
    endif
endif
endfor
until error = true or nothing changed

```

Algorithm 6.4: Incorporation of Upper Bound Constraints

6.10.5 Valid Workflow Executions

At the end of the build-time calculations, there are at least two (possibly not distinct) valid workflow executions (schedules). These executions are obtained if all activities complete at their E-values or their L-values. There may be other valid

combinations of occurrence completion times within (E, L) ranges. We suggest that a timed graph *satisfies* a constraint if the executions in which all occurrences complete at their E- or L-values are valid with respect to this constraint. In addition, by examining the L-values of an occurrence, one can determine if there is a path containing this activity that may lead to time error during process execution. In particular, if all L-values of an activity are greater than their corresponding E-values, there are execution paths containing this activity that are likely to avoid time violations. However, if some L-values are less than their corresponding E-values, then there are paths that may lead to time violations [Eder and Panagos, 2000].

6.11 Example for Computing the TWG

In this Section, by means of an example we will show how our time management technique works. Starting with the workflow of Figure 4.2, we show step by step how our technique is applied.

The initial step is to build the timed workflow graph by applying the algorithm 6.2 and 6.3. Figure 6.10 shows the result after considering structural and lower bound constraints.

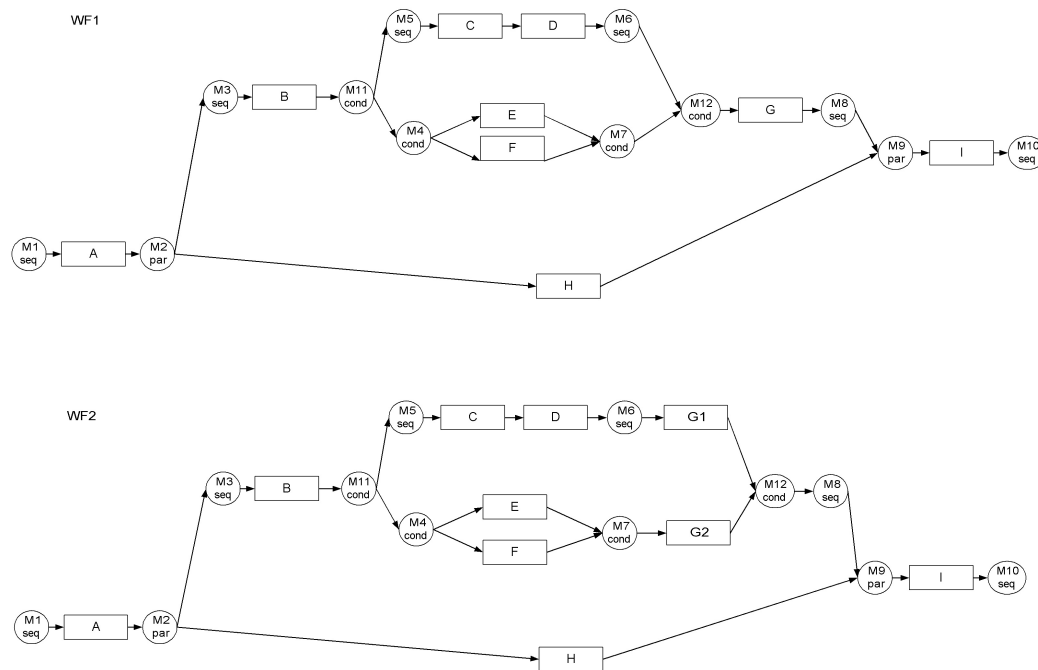


Figure 6.15: Example Workflow - Transformation Step 1 and 2

The next step is to incorporate upper constraints in the timed workflow graph by applying the algorithm 6.4. Here we have two constraints $ubc(B, D, 20)$ and $ubc(C, G, 15)$ that need to be incorporated into the timed graph shown in Figure 6.10. If we incorporate $ubc(B, D, 20)$ first, then $D.L^{wc}$ becomes 22 and, consequently, $ubc(C, G, 15)$ cannot be satisfied as shown in Figure 6.9.

Due to the unsatisfied constraint $ubc(C, G, 15)$, we transform the workflow according to the operations of Chapter 5 in order to separate the inherent instance type that belongs to activity occurrence C and D . This transformation is depicted in Figure 6.15 and 6.16, where the time information is masked. In the upper part of Fig. 6.15, transformation $WFT - J7$ (Join Coalescing) is employed. In the lower part (WF2), transformation $WFT - J1$ (Join Moving Over Activity Occurrence) is applied. In the upper part of Fig. 6.16, transformation $WFT - J2$ (Join Moving Over Seq-Join) is employed and in the lower part (WF4), transformation $WFT - J8$ (Moving Or-Join Over And-Join). After that, transformation $WFT - J1$ and $WFT - J2$ have to be applied.

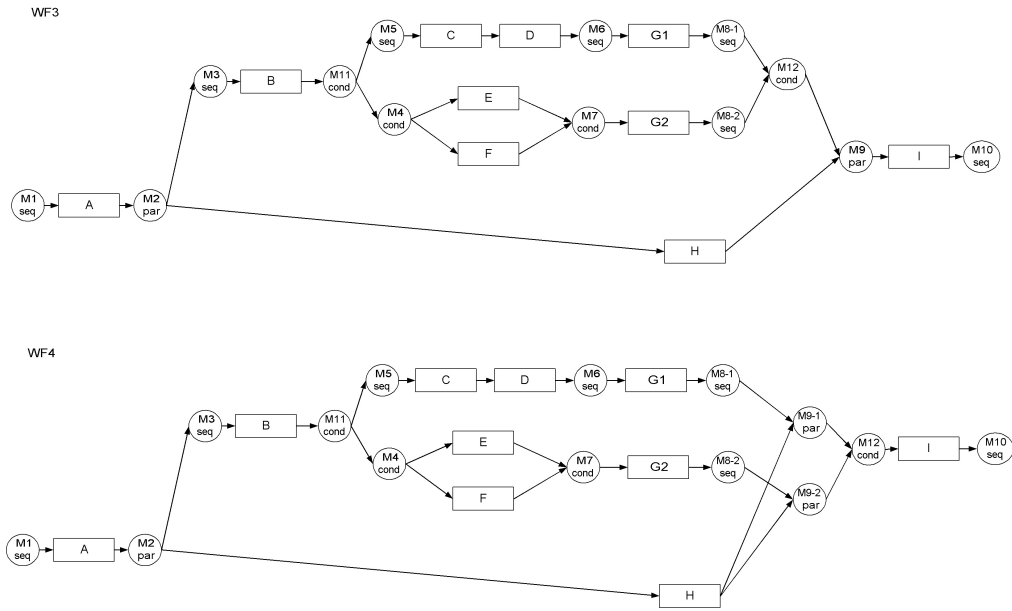


Figure 6.16: Example Workflow - Transformation Step 3 and 4

After the partial unfolding the timed workflow is re-calculated as shown in Figure 6.14.

6.12 Summary

In this Chapter, we presented our technique for modeling, checking, and enforcing temporal constraints in workflow processes containing conditionally and alternatively executed activity occurrences. Our technique distinguishes between time constraints that apply to disparate execution paths and, thus, it avoids the superfluous time constraint violations detected by existing techniques that treat these paths similar to those of unconditionally executed occurrences. In addition, our incorporation of explicit time constraints into the unfolded graph avoids the problem of detecting scheduling conflicts when workflow instances are treated independently of each other.

7

Prototypical Implementation

In this Chapter, we focus on the description of our prototype called *Graphical Workflow Designer (GWfD)* that is developed to support workflow modeling and time management during workflow build-time. The prototype is based on the concepts introduced in the previous Chapters and it is implemented as an autonomous Java application. The main reason to develop the **GWfD** is the proof-of-concept testing in order to determine whether our theoretical concepts are valid and sound, and whether all the promises that have been made about it are met.

7.1 Introduction

The **GWfD** prototype was developed at the University of Klagenfurt, Department of Informatics-Systems, in the course of the “Softwarepraktikum”, winter term 2002, by Stefan Perauer and Robert Sorschag.

In this Chapter, we will describe the functionality of the **GWfD** and show the general practice by means of an example. Here, the implementation aspects and details such as software architecture, the relational database model, database connectivity, etc. are not described unless it is important.

The **GWfD** is implemented in Java and it uses the *JGraph* Swing Component [JGraph] for the graphical workflow representation. Oracle8i serves as database management system (DBMS), where workflows are durably stored. The relational model is gained from our workflow metamodel by applying the steps as proposed e.g. in [Elmasri and Navathe, 2000]. Furthermore, *JDBC* technology provides connectivity to the SQL database and allows us to access the stored workflows.

In the next Sections, a brief overview of the software architecture and the provided functionalities of the **GWfD** is given.

7.1.1 *GWfD* Architecture

The demands on the *GWfD* architecture were primarily system independency, persistent data management, modularity, extensibility and simplicity. System structuring is accomplished by a three layered architecture as shown in Figure 7.1. We use the recommended architectural design pattern *Model-View-Controller* (MVC) as the blueprint for our interactive prototype.

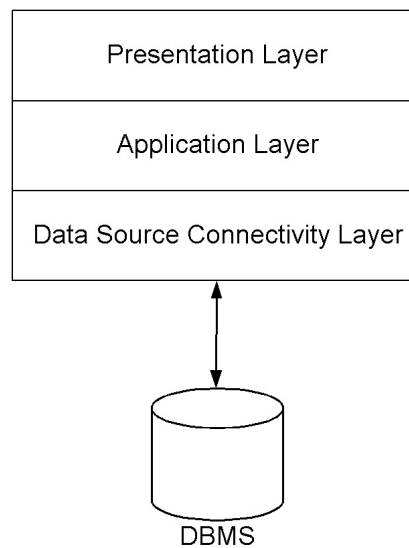


Figure 7.1: *GWfD* Architecture

The three layers of *Data Source Connectivity*, *Application Layer* and *Presentation Layer* are organized in a layered architecture, as, for instance, proposed in [Garlan and Shaw, 1993]. Each layer provides services to the layer above and uses services provided by the layer below.

- **Data Source Connectivity Layer:** This layer represents the API to the data source relevant for the *GWfD*. A data sources may be a database or an XML file.
- **Application Layer:** The major task of this layer is to create, modify, and delete workflow specifications, deduce workflow models from the specification, and perform time calculations and transformations on the workflow model. Then, the result is passed to the presentation layer.
- **Presentation Layer:** This layer is responsible for the visualization of workflow specifications and models. It should be able to present the results in an intuitively graphical way with the usual functionalities.

7.2 GWfD Functionalities

The *GWfD* tool provides functionalities that enable the workflow designer to maintain workflow engineering. These functionalities comprises the following:

■ Workflow specification

- *Creation:* building up a new workflow specification that includes the creation of complex/elementary activities and occurrences as well as the corresponding transitions.
- *Modification:* insertion, deletion and modification of complex/elementary activities and/or occurrences.
- *Deletion:* deletion of a workflow specification or model.

■ Workflow model deduction

Derives a workflow model from the current workflow specification. It is possible to derive multiple workflow models.

■ Workflow transformation

Depending on the context of the occurrences considered, all possible transformations are proposed. Choosing a transformation operation, the workflow model is modified accordingly.

■ Workflow time calculation

E- and L-values for every workflow element are calculated and time constraints are incorporated.

■ Time constraint definition

Time constraints (lbc, ubc) between two appropriate workflow occurrences can be defined and activated or deactivated.

■ Options

- Data Source (DBMS) selection
- Appearance of GUI
- Color setting for graphical model elements

7.3 The Workflow Designer Tool *GWfD*

The *GWfD* tool is described by means of an example. In that way our methodology is demonstrated step by step and depicted by screenshots. The main parts of the *GWfD* are described in the following Sections. A complete description of our workflow design tool can be found in [Perauer and Sorschag, 2003].

7.3.1 The Main Window

The *Main Window* of *GWfD* consists of a menu, a virtual desktop with a minimized specification and model window, a time constraint list (presently minimized) and a command bar (control panel or toolbar) that is placed below the menu. A screenshot of the *Main Window* is shown in Figure 7.2.

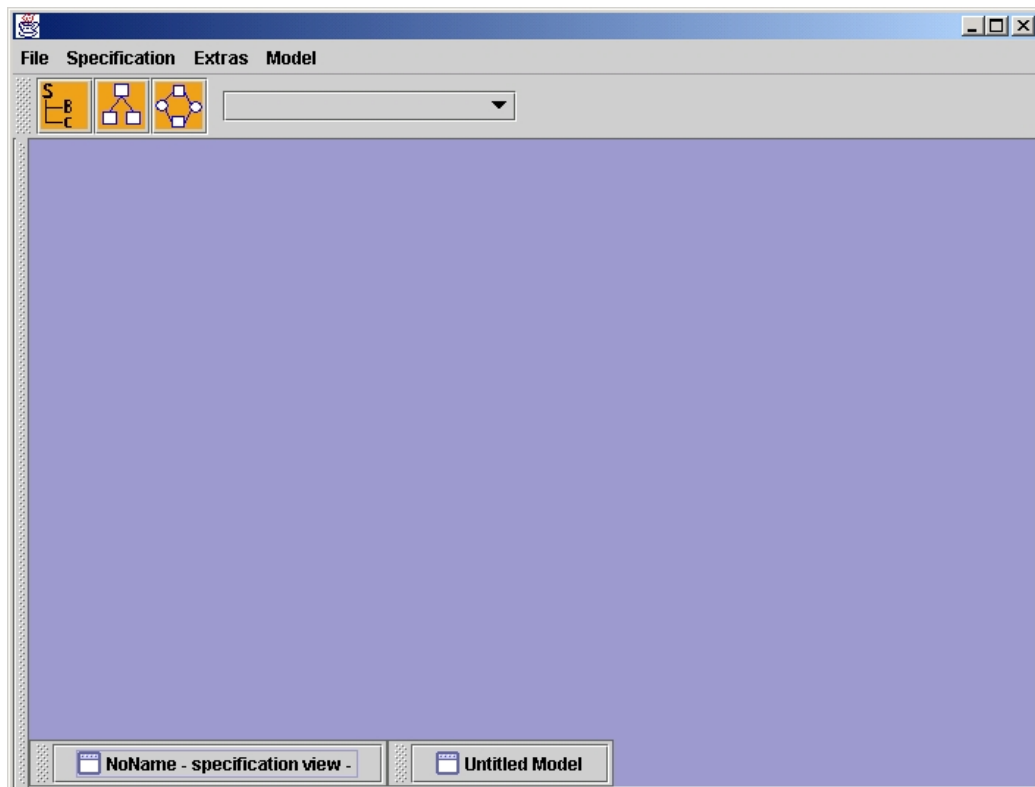
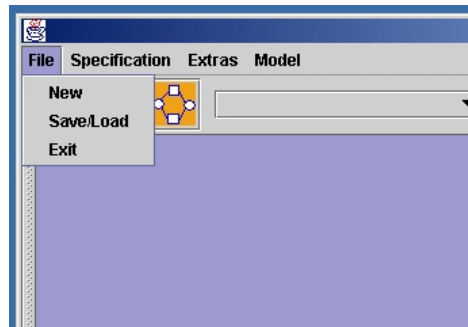


Figure 7.2: Main Window of the *GWfD*

7.3.1.1 The File Menu

The *File* menu offers the usual functionality *New* to generate a new workflow, *Load/Save* to load or store a workflow from/to a data source, and *Exit* to close the application. Figure 7.3 depicts the open *File* menu.

Figure 7.3: File Menu of *GWfD*

7.3.1.2 The Load/Save Window

If the item *Load/Save* in the *File* menu is selected, then the *Load/Save Window* is opened (see Figure 7.4).

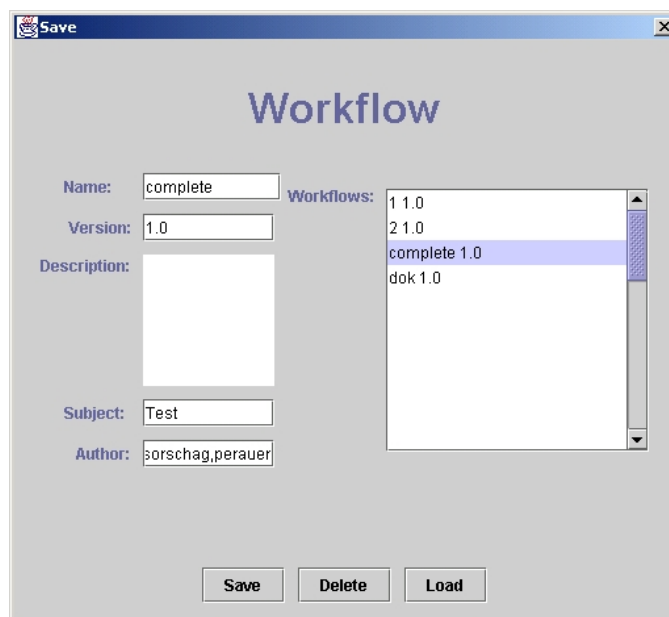


Figure 7.4: Load/Save Window

The list component (list panel) on the right itemizes the workflows stored in the specified data source. If we click on one entry in the list panel, then the corresponding information such as name, version, author, etc. of the selected workflow is shown on the left. We can load or delete the workflow by clicking on the button *Load* or *Delete*. If a workflow is already loaded, then we can store it by clicking on the button *Save*.

7.3.1.3 The Extra Menu

Before we can load a workflow from a data source, we have to set parameters such as the data source location.

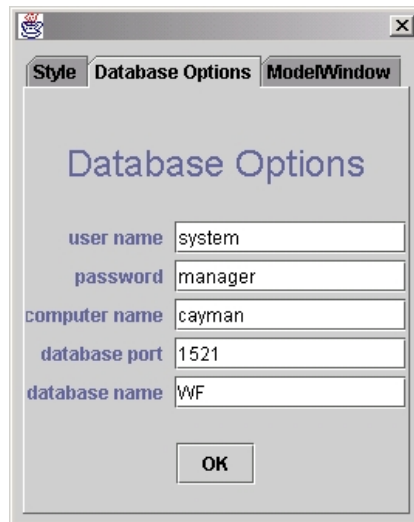


Figure 7.5: Database Location Options

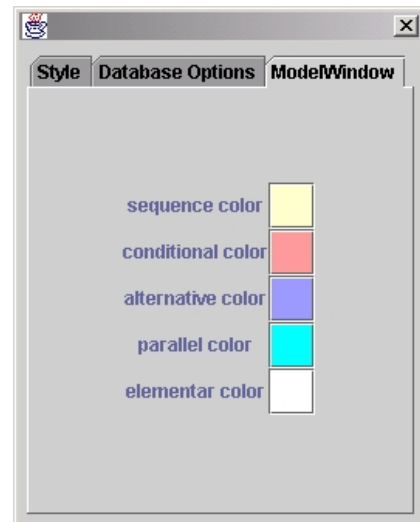


Figure 7.6: Control Element Background Color Options

The *Extra* menu has only the *Option* item. If it is activated by clicking on it, then the *Option* window appears. It contains three tabs which are (i) *Style* to change appearance of the GUI, (ii) *Database Options* to specify the location of the database, and (iii) *ModelWindow* to set the color for model control elements (e.g. the background color of all and-split and and-join elements is set to light green).

Figure 7.5 illustrates the *Database Options* pane with the oracle database location parameters necessary for the JDBC driver. Here, user name, password, database server name, port id and database name are given. Figure 7.6 depicts the *ModelWindow* pane where color setting parameters for the control element background can be changed.

7.3.2 The Specification Editing Window

A workflow can be created or modified by means of the *Specification Editing Window*, which appears by clicking on the leftmost button in the command bar or by opening the *Specification* menu and clicking on the *Edit* item. If no workflow is loaded, then the window is blank; otherwise the loaded workflow elements (activities and occurrences) are listed in the right list panel as it is depicted in Figure 7.7, for example.

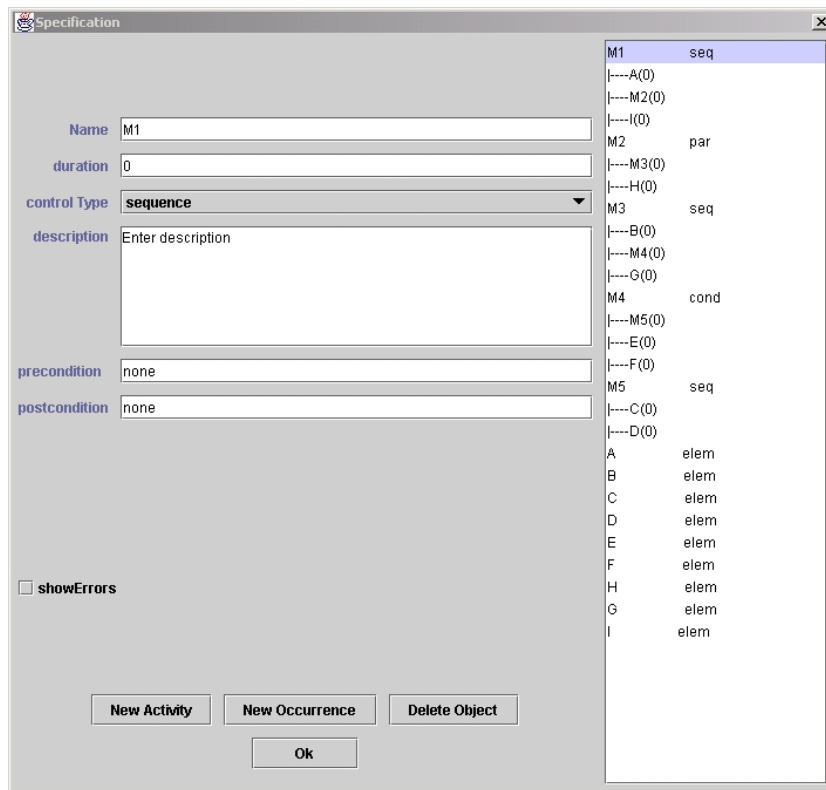


Figure 7.7: Specification Editing Window (loaded Workflow)

In order to specify a workflow, we firstly define the elementary and complex activities. Therefore, we click on the *New Activity* button and the first entry in the list element on the right appears. Then we type in the name, the duration and further information on the given text input fields. After that we choose the type of activity by clicking on the *Control Type* combo-box and assign the wanted activity type. For a complex activity, we choose either *sequence*, *parallel*, *conditional* or *alternative*, and for an elementary activity, we choose *elementar*. To create the next activity, we are clicking on the *New Activity* button again and type in the necessary information.

Occurrences are created in the context of a particular complex activity. This means that we first have to click on a complex activity in the list element, and afterwards we need to click on the *New Occurrence* button. Beneath the complex activity in the list element, a new entry appears that begins with the string “|----”. Similar as for activities, we type in the name, the duration and further information at the given text input fields. Here, the occurrence must have the same name as the corresponding activity. Hence, if we want to create an occurrence of

activity A, the occurrence must be labeled as A. The order of occurrences is significant for complex activities of type *sequence*, because this order represents the execution order and therefore; the transitions are defined accordingly.

Workflow elements can be deleted by selecting it in the list element and clicking the *Delete* button afterwards. If all workflow elements are defined, then the *OK* button has to be clicked. This initiates checking the correctness of the workflow specification, as specified by the integrity constraints in Section 4.4.5, and emerging errors are displayed.

7.3.3 The Specification and Model Window

After we have loaded or created a workflow specification, the *Specification and Model Window* illustrates the workflow as depicted in Figure 7.8, for example.

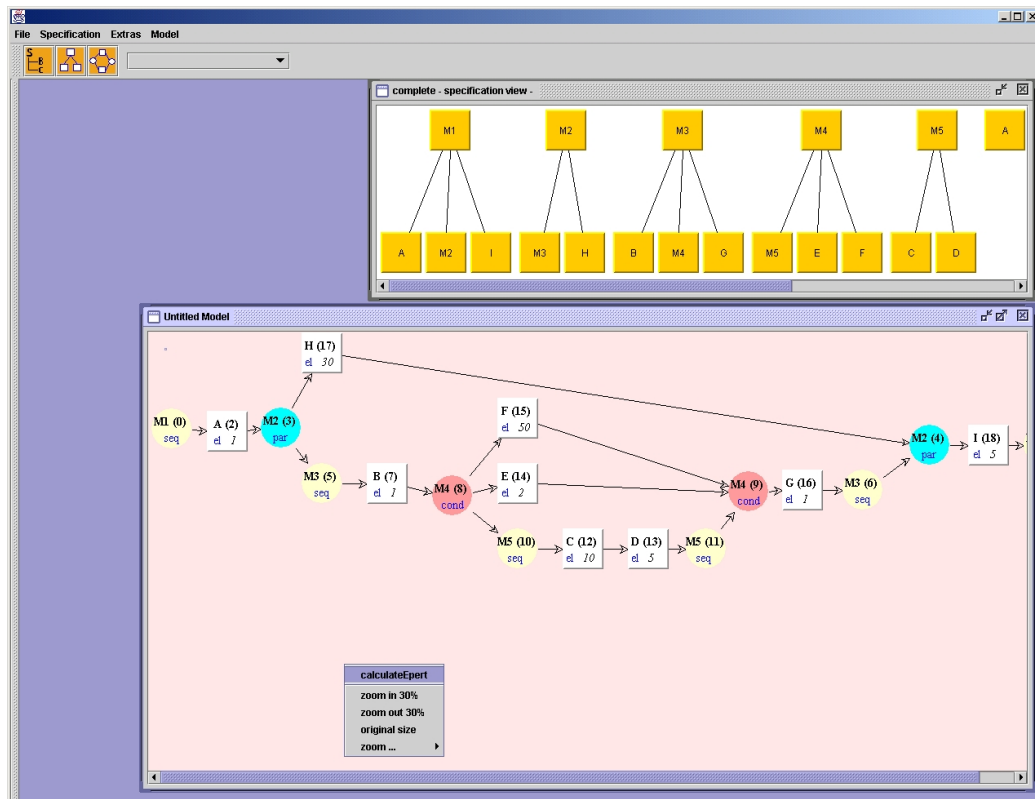


Figure 7.8: Specification and Model Window

The upper window, called Specification Window, represents the workflow specification in graphical form. The lower window, called Model Window, reflects the derived workflow model (cf. Section 3.7). We call this workflow model

master workflow model. Our master workflow model, which can be recognized by the pink window background, allows no modifications (e.g. transformations) of the workflow model. Therefore we can derive (any number of) workflow models by clicking on the *Model* menu and selecting the *GenerateChildModel* item. Such workflow model windows (also called *Child-Model Window*) are identified by the white window background. The nodes in the workflow model windows depict the name and the type of the represented activity or occurrence. Only in elementary activity occurrences, its duration is depicted by numbers in italics.

There are some functionalities concerning the master workflow window that are partially described in the following.

7.3.3.1 Zooming

The *GWfD* supports a zooming feature through which to zoom the display. When the right mouse button is clicked at the desktop, a context menu is displayed (see Figure 7.9). Here we can repeatedly zoom-in and zoom-out in steps of 30%. There are also predefined zooming factors available.

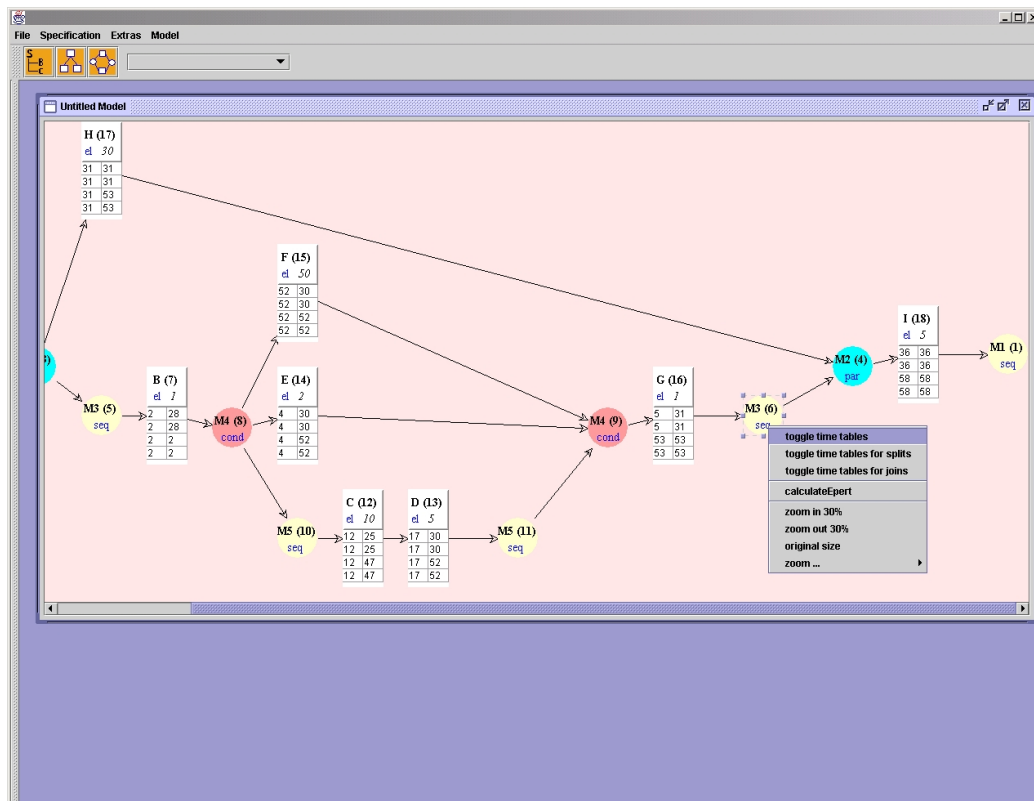


Figure 7.9: Model Window with Context Menu and visible E/L-Values

7.3.3.2 Calculating Time Values

As described in Chapter 6 the E-values and L-values can be calculated for our workflow models. For that purpose, we invoke the context menu and click on the item *CalculateEpert*. After that we can fade in and fade out these time values by making a right mouse click on the regarding nodes. In Figure 7.9, the time values for all elementary occurrences are faded in.

7.3.4 Time Constraint List

The *Time Constraint List* is located on the left in the desktop. To make the *Time Constraint List* pane narrower or wider, the divider on the left has to be pointed at. When the pointer changes to a \leftrightarrow , the left mouse button has been hold down as you drag the divider to the left or right.

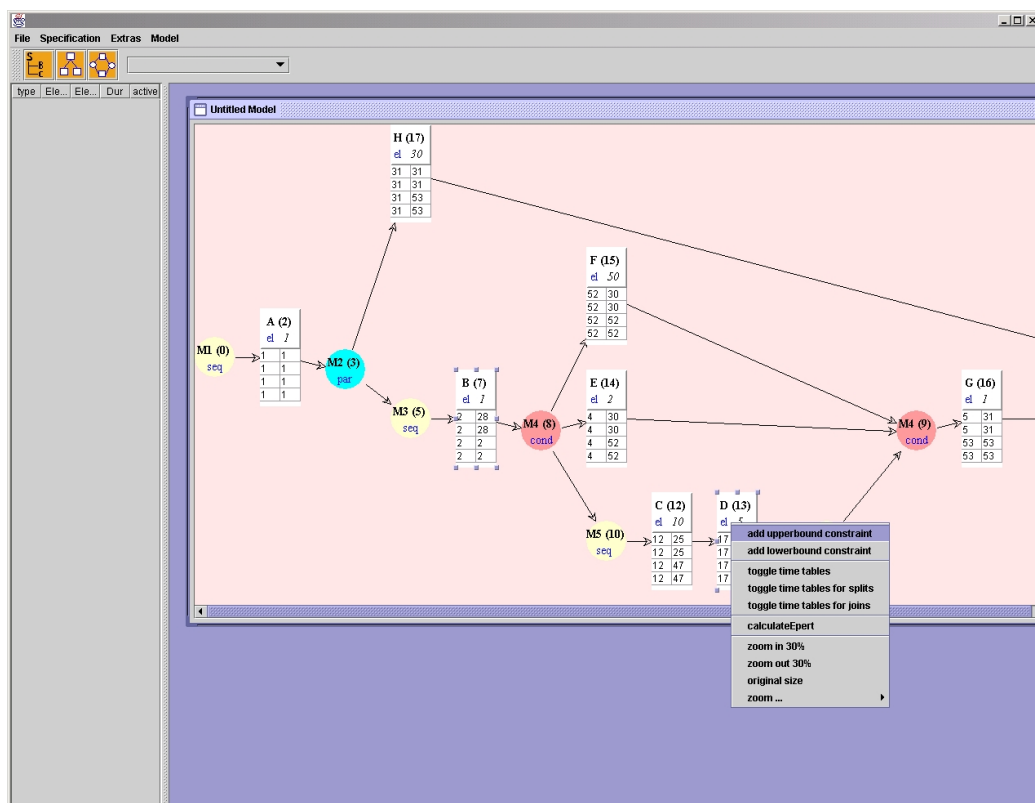


Figure 7.10: Time Constraint List and Activated Context Menu

7.3.4.1 Defining a Relational Time Constraint

Explicit relational time constraints (ubc and lbc) are defined by selecting two activity occurrences. In order to select two elements in the workflow model window at the same time, it is necessary to press and hold the *Ctrl* key and left-click the elements in question. Then a right mouse click on one of the selected items has to be performed to invoke the context menu, which has now two additional items, namely *Add UpperBound Constraint* and *Add LowerBound Constraint* (see Figure 7.10). After one of these items is selected, the user is requested to type in the bound value, and by clicking on the *OK* button, the time constraint is stored and displayed in the *Time Constraint List* if the constraint complies with the integrity constraints.

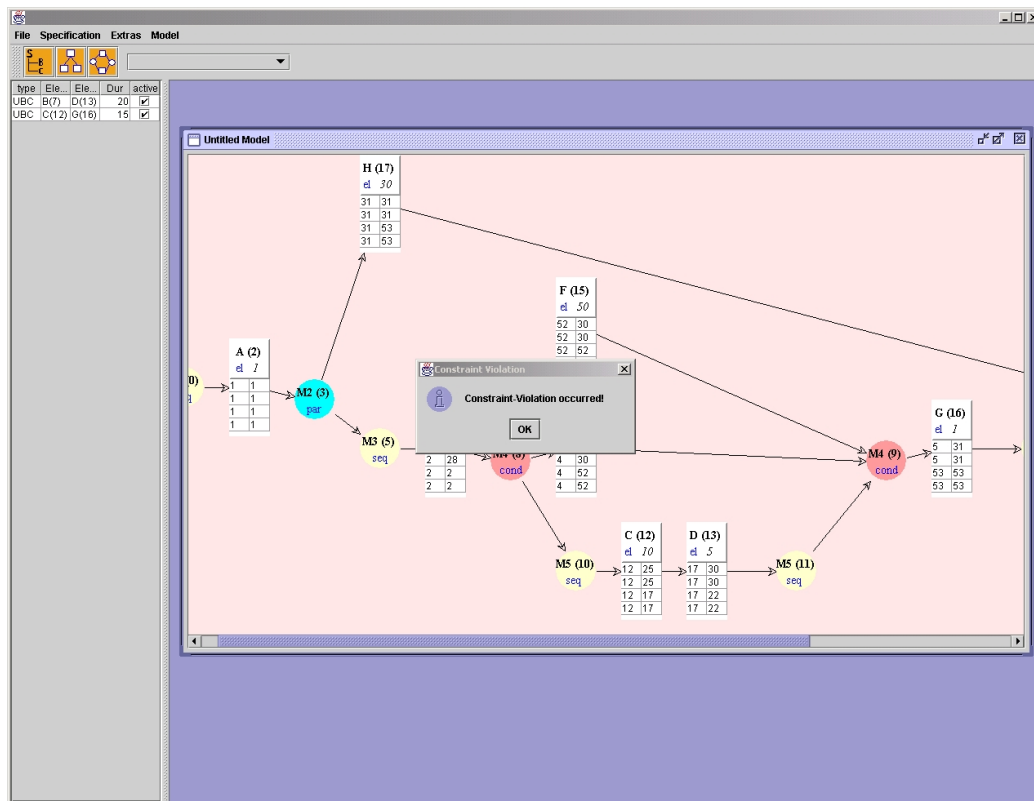


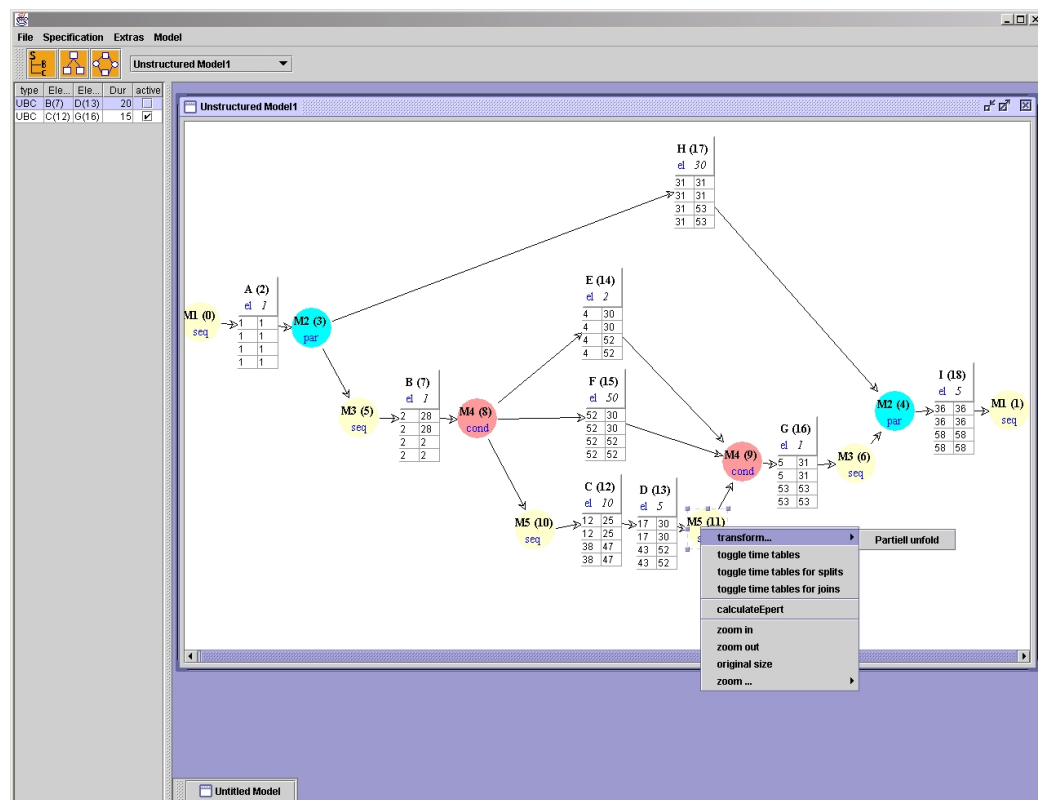
Figure 7.11: Checking Time Constraints

Every entry in the *Time Constraint List* represents an explicit time constraint. The type, the source occurrence, the destination occurrence and the bound of the constraints are shown on the list. In order to improve the handling of our time

management, we added a checkbox to enable (activate) and disable (deactivate) defined constraints. This procedure is preferred to repeatedly deleting and inserting time constraints. Figure 7.11 shows the definition and activation of a ubc time constraints.

7.3.4.2 Checking Time Constraints

Time constraints are checked during the computation of the E-values and L-values as described in Section 7.3.3.2. If a constraint violation occurs, a dialog box with an error message appears (see Figure 7.11). We are now aware, that at least one constraint cannot be satisfied, but we do not know which one. So we can activate and deactivate the time constraints to find a maximum set of satisfied time constraints or to check each time constraint individually. For each unsatisfied time constraint, we perform the transformation operation *Unfold* as described below.

Figure 7.12: Child-Model Window of the *GWfD*

7.3.5 The Child-Model Window

As mentioned above we designed our prototype to make transformation operations possible only in *Child-Model Windows*. In a *Child-Model Window* illustrated in

Figure 7.12, the workflow model is represented in the same way as in the master model window.

7.3.5.1 Applying Transformations

In order to separate the intrinsic instance types in a workflow model, we apply the partial unfold transformation operation. Therefore, we have to specify the instance type to be separated. As illustrated in Figure 7.12, we select an adjacent predecessor of a *cond-join* and invoke the context menu by a right mouse click on the selected predecessor. The first item in the context menu is “*Transform ...*”. This item has to be chosen. After that, a list of all possible transformation operations in this context appears, which in this case could only be the *Partial Unfold* operation. If this operation is selected, the workflow model is accordingly modified and the graphical representation is updated, as we can see in Figure 7.13.

If we perform the recalculation of the E- and L-values, we can see that both time constraints are now satisfied.

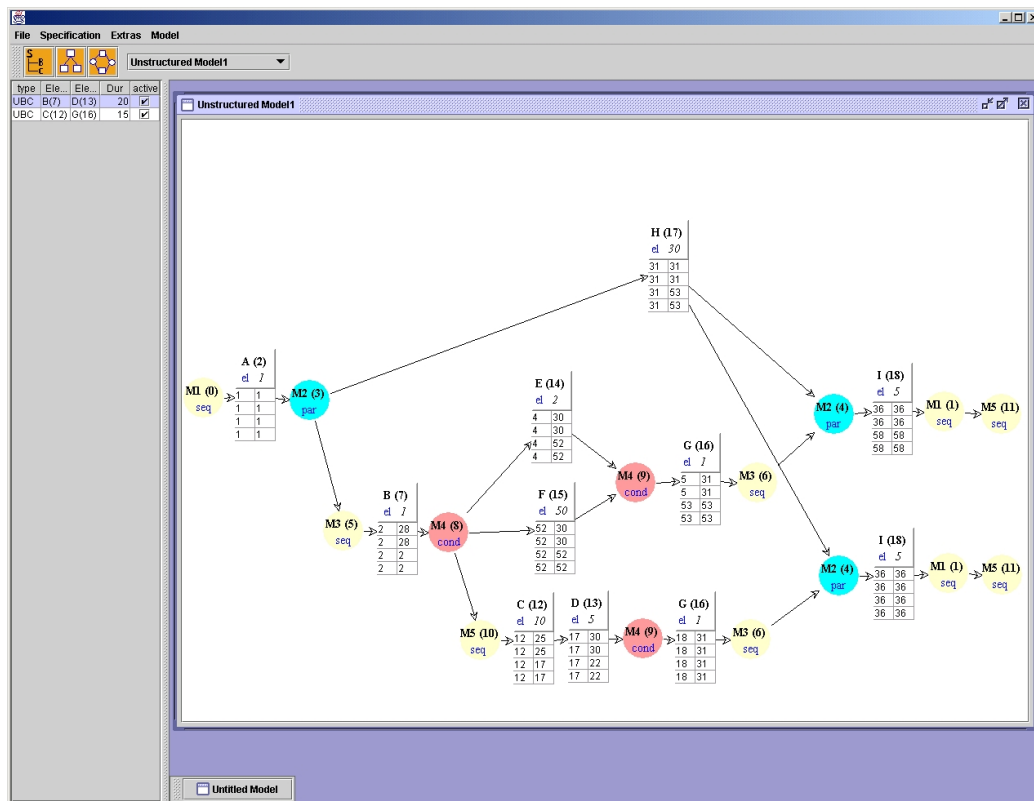


Figure 7.13: Partial Unfolded Workflow Model

There are some other transformations implemented, as described in Chapter 5. Depending on which workflow elements are selected, our prototype proposes all possible transformation operations. In Figure 7.14, for instance, we perform the transformation *Join Moving over Activity*. Therefore, we have to select the *cond-join* and its successor. Then we invoke the context menu by a right mouse click on the selected successor or *cond-join*. The further steps are similar to the ones already described above.

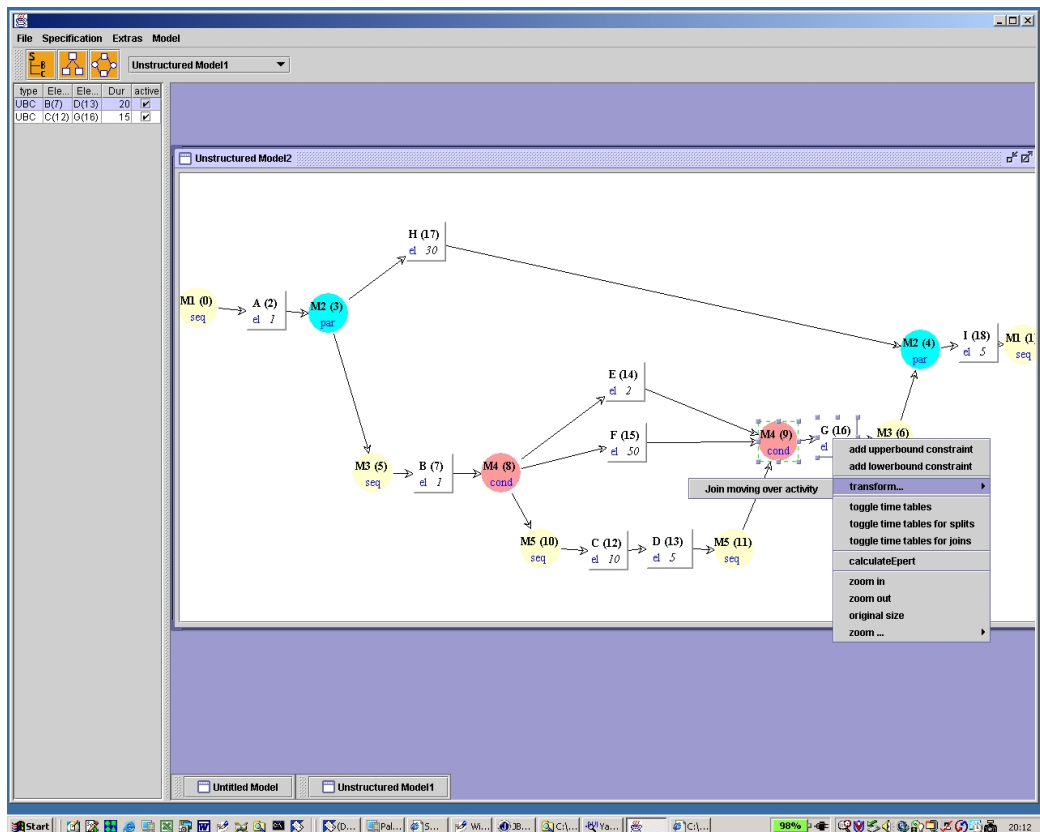


Figure 7.14: Join Moving over Activity

The result is shown in Figure 7.15. A detailed description of applying workflow transformations in *GWfD* can be found in [Perauer and Sorschag, 2003].

7.4 Summary

The algorithms for incorporating explicit time constraints in timed workflow graphs with (partial) unfolding were implemented in a prototype for an extended work-

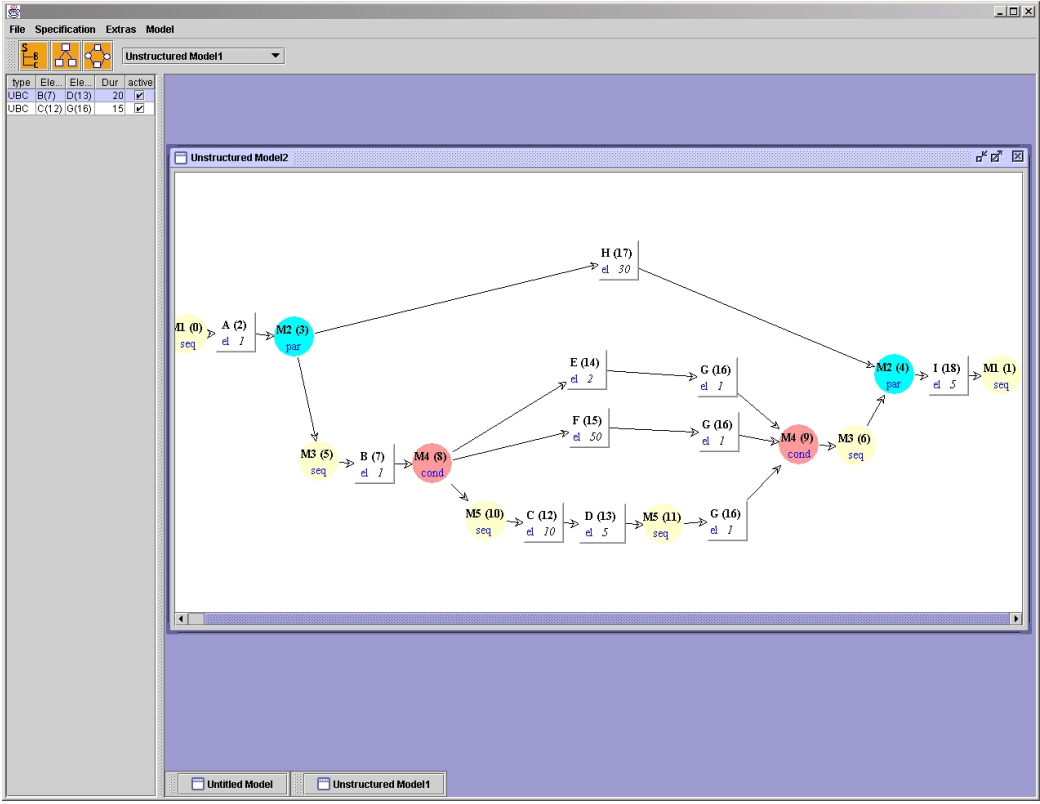


Figure 7.15: Result of Join Moving over Activity of Fig. 7.14

flow design tool. The prototype accepts workflow descriptions in the workflow definition language of the workflow system Panta Rhei [Eder et al., 1997b], extended with explicit time constraints. The algorithms for unfolding and partially unfolding workflow graphs as well as the algorithms for computing timed workflow graphs and incorporating explicit time constraints into these timed graphs are defined by these process definitions.

8

Conclusions and Further Work

In this Chapter we conclude this dissertation by summarizing our contributions and discussing directions for future work.

8.1 Conclusions

Time is a fundamental concept of the universe. We can define time e.g. as a structure of (simultaneous) events on the level of elementary processes. Every elementary process runs on a time scale and, therefore, time plays an important role in workflows.

The main objective of this thesis is to investigate time aspects in workflow management systems and to propose a comprehensive framework for time management.

In this thesis we presented a new approach to time management in workflow systems. After a brief overview of workflow technology, a sophisticated workflow model for control structure oriented processes as well as for graph based processes is presented. Important aspects of our workflow model are the elaborated hierarchical composition supporting re-use of activities by means of *occurrences* and the separation of specification and model level workflow descriptions. Along those lines, we developed a workflow metamodel in UML notation which allows to design and to maintain temporal workflows. Moreover, we defined necessary integrity constraints for this model. To cope with the numerous changes during the lifetime of a workflow model and in order to separate the intrinsic instance types of a workflow model for time management, we developed a set of basic and complex workflow transformations, which do not change the semantics of the workflow. That is why we introduced a new equivalence criterion on workflows and workflow instance types. As main part of this thesis, we addressed the crucial role of time management in workflow processes. In particular, we described how

structural and explicit time constraints can be “captured” during process definition and validated during modeling time. Therefore, the basic temporal concepts used in workflows are introduced and a framework for time modeling is presented. Finally, to demonstrate the approach introduced in this thesis, a graphical workflow designer prototype called **GWfD** has been designed and implemented as an autonomous Java application. **GWfD** is the conceptual modeling and verification tool for workflows designed and developed at the University of Klagenfurt, Austria. Our prototype supports time management and transformation functions during workflow build-time including specification and verification of temporal constraints. Basic functions of **GWfD** are illustrated on workflow examples used throughout this thesis.

Based on the experience made when working on this research project, some final thoughts and conclusions on time management can be drawn:

- Time management in workflows is different from time management in other areas like e.g. production planning and control, artificial intelligence, temporal databases, *etc.*
- Better control of execution times cannot be reached by only introducing a time value attribute for each activity. On the contrary, useful time management means to (i) record possible start and termination times for each activity and the overall process, (ii) define time constraints between activities, (iii) verify the consistency of time constraints, and to (iv) proactively resolve and avoid time constraints violations.
- The verification of temporal consistency is crucial to workflow management in order to guarantee a temporal error-free workflow execution regarding the underlying workflow structure. Therefore, verification during workflow modeling means to determine time constraints that are not satisfied.
- The problem of unnecessary workflow model rejections induced by superfluous time constraint violations is also essential. It can be tackled at maximum by performing workflow transformations.
- A proper time management method during build-time dramatically improves workflow execution by reducing time-driven exceptions, detecting bottlenecks and providing optimization tools in a collaborative environment, testing a workflow before deploying it, communicating with customers, visualizing, and reducing complexity, *etc.*

8.2 Ongoing Work/Further Work

This Section sketches areas of further developments and ongoing work in the field of time management in workflows.

8.2.1 Schedules

The execution of a workflow instance requires the re-computation of the timed graph after the completion of an activity occurrence that is the source of a lower-bound constraint or has a successor that is the source of an upper-bound constraint. These re-computations could be avoided by sacrificing some of the flexibility in the timed graph. The timed graph specifies ranges for activity occurrence completion times so that there *exists* a combination of activity occurrence completion times that satisfies all timing constraints. Run-time re-computation was required because once completion time for finished activity occurrences is observed, not all completion times within the ranges of the remaining activity occurrences continue to be valid [Eder and Panagos, 2000; Eder et al., 1999b].

We define a *schedule* to be a (more restrictive) timed graph in which *any* combination of activity completion times within $[E, L]$ ranges satisfies all timing constraints. In other words, given a schedule no violations of time constraints occur as long as each activity occurrence a finishes on time within the interval $[a.E, a.L]$. Consequently, as long as activity occurrences finish within their ranges, no timed graph re-computation is needed. Only when an activity occurrence finishes outside its range, the schedule for the remaining activities must be recomputed [Eder and Panagos, 2000; Eder et al., 1999b].

Following the schedule definition for every upper-bound constraint $ubc(s, d, \delta)$, $s.E + \delta \geq d.L$ and for every lower-bound constraint $lbc(s, d, \delta)$, $s.L + \delta \leq d.E$ the reverse is also true as well, i.e. the timed graph that satisfies these properties is a schedule. From the way we compute E- and L-values for the activity occurrences in a timed workflow graph, the E- and L-values already qualify as schedules (see Section 6.10.5). Consequently, when every workflow activity occurrence finishes execution at its E-value, there is no need to check for time constraint violations. The same is true when activities finish execution at L-values [Eder and Panagos, 2000; Eder et al., 1999b].

Regarding the scheduling problem, there is plenty literature (see [Suresh and Chaudhuri, 1993] for a good survey) [Bettini et al., 2000].

The development of algorithms for computing schedules with various characteristics is the subject of current research (cf. [Eder et al., 2003; Ninaus, 2002]).

8.2.2 Generalizing to Temporal Reasoning with Multiple Granularities

It is surprising that there is hardly any literature on the subject of temporal reasoning with multiple granularities [Bettini et al., 2002]. In most cases, either the multiple granularities are not considered (e.g. [Dechter et al., 1991]), or it is assumed that constraints in terms of multiple granularities can be equivalently translated in terms of a single granularity. However it is not always possible to convert a temporal distance among two events in terms of one granularity into one in terms of another granularity. Therefore, this work can be extended to multiple time granularities as considered above.

8.2.3 Relaxing the Restrictive Workflow Structure

The temporal workflow graphs in Section 3.5.2 form a subclass of those considered “well-structured”. A topic of further work is to relax the restriction on the workflow structure to arbitrary workflow networks.



Bibliography

G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionalities and limitations of current workflow management systems. Research report, IBM Almaden Research Center, 1997.

[web document.](#) [xvii](#), [10](#), [11](#), [12](#), [86](#)

Jörg Becker, Michael Rosemann, and Christoph von Uthmann. Guidelines of business process modeling. In Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors, *Business Process Management, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 30–49. Springer, 2000.

[web document.](#) [30](#)

C. Bettini, C.E. Dyreson, W.S. Evans, R.T. Snodgrass, and X.S. Wang. A glossary of time granularity concepts. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases: Research and Practice, State-of-the-Art Survey*, volume 1399 of *LNCS*, page 406413. Springer, 1998.

[web document.](#) [92](#)

C. Bettini, X. S. Wang, and S. Jajodia. Free schedules for free agents in workflow systems. In *Proc. of 7th International Workshop on Temporal Representation and Reasoning (TIME2000)*, Cape Breton, Canada, pages 31–38. IEEE Computer Society, 2000.

[web document.](#) [2](#), [89](#), [141](#)

Claudio Bettini, X. Sean Wang, and Sushil Jajodia. Temporal reasoning in workflow systems. In Ahmed K. Elmagarmid, editor, *Distributed and Parallel Databases, An International Journal*, volume 11, pages 269–306. Kluwer Academic Publishers, May 2002.

[web document.](#) [2](#), [85](#), [86](#), [89](#), [92](#), [94](#), [142](#)

Anthony J. Bonner. Workflow, transactions, and datalog. In *Proc. 18th ACM SIGACT-SIGMODSIGART Symposium on Principles of Database Systems*,

Philadelphia, pages 294–305, 1999.
[web document.](#) 19

Christoph Bussler. Workflow instance scheduling with project management tools. In Roland Wagner, editor, *Ninth International Workshop on Database and Expert Systems Applications (DEXA'98), Vienna, Austria, August 24-28, 1998, Proceedings*, pages 753–758. IEEE Computer Society, 1998.
[web document.](#) 2, 87

F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual modeling of workflows. In M. P. Papazoglou, editor, *Object-Oriented and Entity-Relationship Modelling, 14th International Conference, OOER'95, Gold Coast, Australia, December 12-15, 1995, Proceedings*, volume 1021 of *Lecture Notes in Computer Science*, pages 341–354. Springer, 1995.
[web document.](#) 31, 45

F. Casati, B. Pernici, G. Pozzi, G. Sanchez, and J. Vonk. *Database Support for Workflow Management: The WIDE Project*, chapter Conceptual workflow model. Kluwer Academic Publishers, 1999. 88

Fabio Casati, Stefano Ceri, Barbara Pernici, and Giuseppe Pozzi. Deriving active rules for workflow enactment. In R. Wagner and H. Thoma, editors, *Proceedings of the Seventh International Conference on Database and Expert Systems Applications (DEXA '96, Zurich, Switzerland, Sept.)*, pages 94–115, 1996a.
[web document.](#) 19

Fabio Casati, Stefano Ceri, Barbara Pernici, and Giuseppe Pozzi. Workflow evolution. In Bernhard Thalheim, editor, *Conceptual Modeling - ER'96, 15th International Conference on Conceptual Modeling, Cottbus, Germany, October 7-10, 1996, Proceedings*, volume 1157 of *Lecture Notes in Computer Science*, pages 438–455. Springer, 1996b.
[web document.](#) 2, 45

Jan Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. In *ACM Transactions on Database Systems*, volume 20, pages 149–186, 1995.
[web document.](#) 36

J. Clifford and A. Rao. A simple, general structure for temporal domains. In C. Rolland, F. Bodart, and M. Leonard, editors, *Temporal Aspects in Information Systems, Proceedings of the IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems, Sophia-Antipolis, France, 13-15*

- May, pages 17–28. North-Holland/IFIP, Amsterdam, The Netherlands, 1987.
92
- Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. In *Communications of the ACM, Volume 35, Issue 9 (September 1992) Special issue on analysis and modeling in software development*, volume 35, pages 75–90. ACM, 1992.
web document. 30
- P. Dadam, M. Reichert, and K. Kuhn. Clinical workflows the killer application for processoriented information systems. In *4th International Conference on Business Information System (BIS 2000), Poznan, Poland*, pages 36–59, 2000.
web document. 2, 88, 104, 105
- R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. In *Artificial Intelligence*, volume 49, pages 61–95, 1991.
web document. 93, 142
- Christian Dorninger, Otto Janschek, Erlefried Olearczick, and Hans Röhrenbacher. *PPS Produktionsplanung und -steuerung - Konzepte, Methoden und Kritik*. Ueberreuter, Wien, 1990. 105
- J. Eder, W. Gruber, and E. Panagos. Temporal modeling of workflows with conditional execution paths. In Mohamed T. Ibrahim, Josef Küng, and Norman Revell, editors, *Database and Expert Systems Applications, 11th International Conference, DEXA 2000, London, UK, September 4-8, 2000, Proceedings*, volume 1873 of *Lecture Notes in Computer Science*, pages 243–253. Springer, 2000.
web document. 2, 15, 68, 75, 77, 79, 86, 89, 93, 104, 106, 109, 110, 113
- J. Eder and W. Liebhart. Workflow transactions. In Peter Lawrence, editor, *Workflow Handbook 1997*. John Wiley & Son Ltd, 1 edition (January 29, 1997), 1997.
web document. 88
- J. Eder and E. Panagos. Managing Time in Workflow Systems. In L. Fischer, editor, *Workflow Handbook 2001*, pages 109–132. Future Strategies INC. in association with Workflow Management Coalition, 2000.
web document. 2, 86, 90, 91, 92, 93, 94, 95, 96, 97, 104, 109, 110, 113, 119, 141
- J. Eder, E. Panagos, H. Pozewaunig, and M. Rabinovich. Time management in workflow systems. In W. Abramowicz and M.E. Orłowska, editors, *Business Information Systems, 3rd International Conference, BIS'99, Poznan, Poland*,

- April 14-16, 1999, Proceedings*, pages 265–280. Springer, 1999a.
web document. 24, 86, 89, 104, 106
- J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In Matthias Jarke and Andreas Oberweis, editors, *Advanced Information Systems Engineering, 11th International Conference, CAiSE'99, Heidelberg, Germany, June 14-18, 1999, Proceedings*, volume 1626 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 1999b.
web document. 30, 86, 89, 92, 93, 94, 96, 97, 106, 109, 110, 113, 117, 141
- J. Eder, H. Pozewaunig, and W. Liebhart. Timing issues in workflow management systems. Technical report, Universität Klagenfurt, Institut für Informatik-Systeme, 1997a. 88
- Johann Eder, Herbert Groiss, and Walter Liebhart. The workflow management system panta rhei. In Asuman Dogac, Leonid Kalinichenko, M. Tamer Ozsu, and Amit Sheth, editors, *Advances in Workflow Management Systems and Interoperability, 1997*, pages 129–144, 1997b.
web document. 14, 16, 18, 20, 137
- Johann Eder and Wolfgang Gruber. A meta model for structured workflows supporting workflow transformations. In Yannis Manolopoulos and Pavol Navrat, editors, *Sixth East-European Conference on Advances in Databases and Information Systems, September 8-11, 2002, Bratislava, Slovakia*, volume 2435 of *Lecture Notes in Computer Science*, pages 326–339. Springer, September 2002.
web document. 14, 15, 16, 17, 22, 29, 30, 32, 47, 48, 51, 86
- Johann Eder and Walter Liebhart. The workflow activity model WAMO. In Steve Laufmann, Stefano Spaccapietra, and Toshio Yokoi, editors, *Cooperative Information Systems, 3rd International Conference, CoopIS, Vienna, Austria, May 9-12, 1995, Proceedings*, pages 87–98, 1995.
web document. 16
- Johann Eder, Michael Ninaus, and Horst Pichler. Personal schedules for workflow systems. In *International Conference on Business Process Management, Eindhoven, The Netherlands, June 26-27, 2003*, 2003. 35, 141
- Johann Eder, Georg E. Olivotto, and Wolfgang Gruber. A data warehouse for workflow logs. In Y. Han, S. Tai, and D. Wikarski, editors, *Proceedings of the International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002), September 18-20, 2002, Beijing, China*, volume 2480 of *Lecture Notes in Computer Science*, pages 1–15. Springer,

September 2002.

web document. 32, 90

Johann Eder and Horst Pichler. Duration histograms for workflow systems. In *Proceedings of the Working Conference on Engineering Information Systems in the Internet Context (IFIP TC8/WG8.1), September 25-27, 2002, Kanazawa, Japan*, pages 239–253. Kluwer Academic Publishers, 2002. 25, 93, 106

Horst A. Eiselt and Helmut von Frajer. *Operations Research Handbook - Standard Algorithms and Methods with Examples*. Walter de Gruyter, Berlin, 1977. 105

Clarence A. Ellis and Karim Keddara. A Workflow Change Is a Workflow. In Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors, *Business Process Management, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 201–217. Springer, 2000. 45

Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, Reading, MA, 3rd edition edition, 2000. 123

Flo. *Collaborative workflow system for the way people work*. TeamWare Flow, P.O. Box 780, FIN-00101, Helsinki, Finland. 85, 93

R. Les Galloway. *Principles of operations management*. Routledge, London, New York, 1993. 105

David Garlan and Mary Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume 2, pages 1–39. World Scientific Publishing Company, 1993.
web document. 124

Dimitrios Georgakopoulos, Mark F. Hornick, and Amit P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
web document. 2, 7, 10, 85

Andreas Geppert and Dimitris Tombros. Logging and post-mortem analysis of workflow executions based on event histories. In Andreas Geppert and Mikael Berndtsson, editors, *Rules in Database Systems, Third International Workshop, RIDS '97, Skövde, Sweden, June 26-28, 1997, Proceedings*, volume 1312 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 1997.
web document. 90

- Fred Glover, Darwin Klingman, and Nancy V. Phillips. *Network Models in Optimization and Their Applications in Practice*. John Wiley & Sons, Inc, New York, Chister, et. al., 1992. 105
- I. J. Haimowitz, J. Farley, G. S. Fields, and J. Stillman. Temporal reasoning for automated workflow in health care enterprises. In Nabil R. Adam and Yelena Yesha, editors, *Electronic Commerce, Current Research Issues and Applications [Workshop at NIST, Gaithersburg, Maryland, USA, December 1, 1994]*, volume 1028 of *Lecture Notes in Computer Science*, pages 87–113. Springer, 1996. 88
- A. H. M. ter Hofstede, Maria E. Orlowska, and Jayantha Rajapakse. Verification problems in conceptual workflow specifications. In Bernhard Thalheim, editor, *Conceptual Modeling, 15th International Conference, ER'96, Cottbus, Germany, 7-10 October 1996, Proceedings*, volume 1157 of *Lecture Notes in Computer Science*, pages 73–88. Springer, 1996.
web document. 31
- David Hollingsworth. Workflow management coalition: The workflow reference model, tc-1003, issue 1.1, january 1995, 1995. <http://www.wfmc.org/>,
web document. xvii, 2, 7, 8, 9, 10, 85
- InC. *InConcert, Technical product overview*. XSoft. a division of xerox, 3400 Hillview Avenue, Palo Alto, CA 94304, <http://www.xsoft.com>. 2, 85, 93
- S. Jablonski. Mobile: A modular workflow model and architecture. In *Proc. of Int'l Working Conference on Dynamic Modelling and Information Systems, Nordwijkerhout, 1994*, 1994.
web document. 31
- S. Jablonski and C. Bussler. *Workflow Management*. International Thomson Computer Press, London, 1996. 7, 30
- H. Jasper and O. Zukunft. Zeitaspekte bei der Modellierung und Ausführung von Workflows. In S. Jablonski, H. Groiss, R. Kaschek, and W. Liebhart, editors, *Reihe der Informatik '96*, pages 109–119, 1996. 2, 87
- H. Jasper, O. Zukunft, and H. Behrends. Time Issues in Advanced Workflow Management Applications of Active Databases. In Mikael Berndtsson and Jörgen Hansson, editors, *Proceedings of the First International Workshop on Active and Real-Time Database Systems (ARTDB-95), Skovde, Sweden, 9-11 June 1995*, pages 65–81. Springer, 1996.
web document. 92

- JGraph. The JGraph Project (v.2.1), <http://www.jgraph.com>, 2003. 123
- Gregor Joeris and Otthein Herzog. Managing evolving workflow specifications. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, New York City, New York, USA, August 20-22, 1998, Sponsored by IFCIS, The Intn'l Foundation on Cooperative Information Systems*, pages 310–321. IEEE-CS Press, 1998.
web document. 31, 45
- Gregor Joeris and Otthein Herzog. Towards flexible and high-level modeling and enacting of processes. In Matthias Jarke and Andreas Oberweis, editors, *Advanced Information Systems Engineering, 11th International Conference, CAiSE'99, Heidelberg, Germany, June 14-18, 1999, Proceedings*, volume 1626 of *Lecture Notes in Computer Science*, pages 88–102. Springer, 1999.
web document. 13
- B. Kao and H. Garcia-Molina. Deadline assignment in a distributed soft real-time system. In *Proceedings of the 13th International Conference on Distributed Computing Systems, May 25-28, 1993, Pittsburgh, Pennsylvania, USA*, pages 428–437. IEEE Computer Society Press, 1993a.
web document. 87
- B. Kao and H. Garcia-Molina. Subtask deadline assignment for complex distributed soft realtime tasks. Technical report 931491, Stanford University, 1993b.
web document. 87
- Gerti Kappel, Peter Lang, S. Rausch-Schott, and Werner Retschitzegger. Workflow management based on objects, rules, and roles. In *Data Engineering Bulletin*, volume 18, pages 11–18, 1995.
web document. 19, 20, 31
- Henry A. Kautz and Peter B. Ladkin. Integrating metric and qualitative temporal reasoning. In *National Conference on Artificial Intelligence*, pages 241–246, 1991.
web document. 92
- B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On structured workflow modelling. In Benkt Wangler and Lars Bergman, editors, *Advanced Information Systems Engineering, 12th International Conference, CAiSE'00, Stockholm, Sweden, June 5-9, 2000, Proceedings*, volume 1789 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 1999.
web document. 15, 48, 71

- Brigitte Kolmann. *Zeitmanagement in Workflow Systemen*. Master's thesis, Universität Klagenfurt, 2001. 97
- M. Kradolfer. *A Workflow Metamodel Supporting Dynamic, Reuse-Based Model Evolution*. PhD thesis, Universität Zürich, 2000.
web document. 30, 32
- Narayanan Krishnakumar and Amit P. Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. In Benkt Wangler and Lars Bergman, editors, *Distributed and Parallel Databases*, volume 3, pages 155–186, 1995.
web document. 32
- P. Lawrence. *Workflow Handbook*. John Wiley and Sons, New York, 1997. 2, 85
- F. Leymann and D. Roller. Business process management with flowmark. In *Proceedings of the 39th IEEE Computer Society International Conference, San Francisco, California, February 1994*, pages 230–233, 1994.
web document. 2, 85, 93
- Walter Liebhart. *Fehler- und Ausnahmebehandlung im Workflow Management*. PhD thesis, Universität Klagenfurt, 1998. 16, 32
- H. Lin, Z. Zhao, H. Li, and Z. Chen. A novel graph reduction algorithm to identify structural conflicts. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02), Big Island, Hawaii, 7-10 January, 2002*, volume 9. Computer Society Press, 2002.
web document. 15
- Olivera Marjanovic. Dynamic verification of temporal constraints in production workflows. In *Australasian Database Conference, 31 January - 3 February, 2000, Canberra, Australia*, pages 74–81. IEEE press, 2000.
web document. 88, 93
- Olivera Marjanovic. Methodological considerations for time modeling in workflows. In *Proceedings of the twelfth Australasian Conference on Information Systems, 4-7 December 2001, Coffs Harbour, NSW, 2001*.
web document. 88
- Olivera Marjanovic and Maria E. Orlowska. On modeling and verification of temporal constraints in production workflows. In Xindong Wu and Benjamin W. Wah, editors, *Knowledge and Information Systems, KAIS*, volume 1, pages 157–192. Springer, may 1999a. 23, 88

- Olivera Marjanovic and Maria E. Orlowska. Time management in dynamic workflows. In Yanchun Zhang, Marek Rusinkiewicz, and Yahiko Kambayashi, editors, *Cooperative Database Systems and Applications '99, The Proceedings of the Second International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'99), Wollongong, Australia, March 27-28, 1999*, pages 138–149. Springer, 1999b. 2, 30, 48, 88, 94
- Klaus Meyer-Wegener and Markus Böhm. Conceptual workflow schemas. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems, Edinburgh, Scotland, September 2-4, 1999*, pages 234–242. IEEE Computer Society Press, 1999.
web document. 32
- Martin E. Modell. *A Professional's Guide to Systems Analysis, 2nd Edition*. McGraw-Hill, 1996. 101, 102
- M. Ninaus. Auslastungsberechnungen in probabilistischen Workflow Systemen. Master's thesis, Universität Klagenfurt, 2002. 35, 141
- E. Panagos and M. Rabinovich. Escalations in workflow management systems. In *DART Workshop, Rockville, Maryland, November 1996*, 1996.
web document. 87
- E. Panagos and M. Rabinovich. Predictive workflow management. In *Proceedings of the 3rd International Workshop on Next Generation Information Technologies and Systems, Neve Ilan, Israel, June 1997*, 1997a.
web document. 87
- E. Panagos and M. Rabinovich. Reducing escalation-related costs in WFMSs. In A. Dogac et al., editor, *NATO Advanced Study Institute on Workflow Management Systems and Interoperability*. Springer, 1997b.
web document. 2, 86, 87
- Stefan Perauer and Robert Sorschag. *Workflow Designer Spezifikation und Implementierung (23.3.2003)*. Universität Klagenfurt, ISYS, Softwarepraktikum, 2003. 126, 136
- Susy Philipose. *Operations Research - A Practical Approach*. Tata McGraw-Hill, New Delhi, New York, 1986. 105, 106
- H. Pozewaunig. Behandlung von Zeit in Workflow-Managementsystemen. Modellierung und Integration. Master's thesis, Universität Klagenfurt, 1996. 88, 103

- H. Pozewaunig, J. Eder, and W. Liebhart. *epert: Extending PERT for workflow management systems*. In *First EastEuropean Symposium on Advances in Database and Information Systems ADBIS '97*, 1997.
[web document](#). xviii, 2, 17, 85, 86, 87, 89, 101, 103, 105, 106
- Manfred Reichert and Peter Dadam. ADEPT flex -supporting dynamic changes of workflows without losing control. In *Journal of Intelligent Information Systems*, volume 10, pages 93–129, 1998.
[web document](#). 45
- Peter Z. Revesz. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. In *Theoretical Computer Science*, volume 116, pages 117–149, 1993.
[web document](#). 93
- Wasim Sadiq, Olivera Marjanovic, and Maria E. Orlowska. Managing change and time in dynamic workflow processes. In *International Journal of Cooperative Information Systems (IJCIS)*, volume 9, pages 93–116, 2000.
[web document](#). 32
- Wasim Sadiq and Maria E. Orlowska. Applying graph reduction techniques for identifying structural conflicts in process models. In Matthias Jarke and Andreas Oberweis, editors, *Advanced Information Systems Engineering, 11th International Conference CAiSE'99, Heidelberg, Germany, June 14-18, 1999, Proceedings*, volume 1626 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 1999a.
[web document](#). 15
- Wasim Sadiq and Maria E. Orlowska. On capturing process requirements of workflow based business information systems. In W. Abramowicz and M.E. Orlowska, editors, *Business Information Systems, 3rd International Conference, BIS'99, Poznan, Poland, April 14-16, 1999, Proceedings*, pages 281–294. Springer, 1999b.
[web document](#). 30
- Wasim Sadiq and Maria E. Orlowska. On business process model transformations. In Alberto H. F. Laender, Stephen W. Liddle, and Veda C. Storey, editors, *Conceptual Modeling, 19th International Conference, ER 2000, Salt Lake City, Utah, USA, October 9-12, 2000, Proceedings*, volume 1920 of *Lecture Notes in Computer Science*, pages 267–280. Springer, 2000.
[web document](#). 47, 48, 65
- SAP. Germany. *SAP Business Workflow@OnlineHelp, 1997. Part of the SAP System*. 2

- G. Schmidt. Scheduling models for workflow management. In B. Scholz-Reiter and E. Stickel, editors, *Business Process Modelling*. Springer, 1996. 2, 85
- Eddie Schwalb and Lluís Vila. Temporal constraints: A survey. In *Constraints*, volume 3, pages 129–149. Kluwer Academic Publishers, 1998.
web document. 93
- V. Suresh and D. Chaudhuri. Dynamic scheduling: A survey of research. In *International Journal of Production Economics*, volume 32, 1993.
web document. 141
- Goce Trajcevski, Chitta Baral, and Jorge Lobo. Formalizing (and reasoning about) the specifications of workflows. In Opher Etzion and Peter Scheuermann, editors, *Cooperative Information Systems, 7th International Conference, CoopIS 2000, Eilat, Israel, September 6-8, 2000, Proceedings*, volume 1901 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2000.
web document. 32
- Ultimus. *Workflow suite. Business workflow automation*. 4915 Waters Edge Dr., Suite 135, Raleigh, NC 27606. <http://www.ultimus.com>. 2
- W. M. P. van der Aalst. The application of petri nets to workflow management. In *The Journal of Circuits, Systems and Computers*, volume 8, pages 21–66, 1998.
web document. xvii, 17
- W. M. P. van der Aalst. Generic workflow models: How to handle dynamic change and capture management information? In *Proceedings of the Fourth IF-CIS International Conference on Cooperative Information Systems, Edinburgh, Scotland, September 2-4, 1999*, pages 115–126. IEEE Computer Society Press, 1999.
web document. 45
- W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede, and Bartek Kiepuszeski. Advanced workflow patterns. In Opher Etzion and Peter Scheuermann, editors, *Cooperative Information Systems, 7th International Conference, CoopIS 2000, Eilat, Israel, September 6-8, 2000, Proceedings*, volume 1901 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 2000.
web document. 23, 48
- W. M. P. van der Aalst, Twan Basten, H. M. W. Verbeek, Peter A. C. Verkoulen, and Marc Voorhoeve. Adaptive workflow - on the interplay between flexibility and support. In *International Conference on Enterprise Information Systems*, pages 353–360, 1999. 46, 47, 49

- W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszeski, and A. P. Barros. Workflow patterns (report 2002). Technical report FIT-TR-2002-02, Queensland University of Technology, Brisbane, 2002. (To appear in Distributed and Parallel Databases.),
web document. 13, 14, 23
- Wil van der Aalst and Kees van Hee. *Workflow Management (models, methods and systems)*. MIT Press, Cambridge, London, 2002. 7, 13, 17
- W.M.P. van der Aalst. Exterminating the dynamic change bug - a concrete approach to support workflow change, 2001.
web document. 45
- W.M.P. van der Aalst, K.M. van Hee, and G.J. Houben. Modelling workflow management systems with high-level petri nets. In G. De Michelis, C. Ellis, and G. Memmi, editors, *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50, 1994.
web document. 87
- J. Warmer and A. Kleppe. *The Object Constraint Language*. Addison-Wesley, 1999. 32
- WfMC. Workflow Management Coalition, brussels, belgium: Glossary (issue 1.0): A workflow management coalition specification, november 1994, 1994a.
<http://www.wfmc.org/>. 8
- WfMC. Workflow Management Coalition, brussels, belgium: Glossary (issue 2.0): A workflow management coalition specification, november 1994, 1994b.
<http://www.wfmc.org/>. 8
- WfMC. Workflow Management Coalition, brussels, belgium: Terminology & glossary (issue 3.0): A workflow management coalition specification, document number wfmc-tc-1011, issue 3.0, february 1999, 1999a.
<http://www.wfmc.org/>,
web document. 8, 23, 85, 90
- WfMC. Workflow management coalition interface 1: Process definition interchange process model, technical report wfmc-tc-1016-p, ver. 1.1, october 1999, 1999b. <http://www.wfmc.org/>,
web document. 2, 15, 32
- WfMC. *Workflow Handbook 2001*. Future Strategies Inc., Lighthouse Point, FL, USA, 2000. 7, 8

WfMC. *Workflow Handbook 2002*. Future Strategies Inc., Lighthouse Point, FL, USA, 2001. 7, 8

WfMC. *Workflow Handbook 2003*. Future Strategies Inc., Lighthouse Point, FL, USA, 2002. 7

Carl-Alexander Wichert, Alfred Fent, and Burkhard Freitag. A logical framework for the specification of transactions (extended version). Technical Report MIP-0102, Universität Passau (FMI), 2001.
[web document](#). 19

Dirk Wodtke and Gerhard Weikum. A formal foundation for distributed workflow execution based on state charts. In *ICDT 1997*, pages 230–246, 1997.
[web document](#). 18

J. Leon Zhao and Edward A. Stohr. Temporal workflow management in a claim handling system. In *Work activities coordination and collaboration, International Joint Conference, WACC'99, San Francisco, CA USA, February 22 - 25, 1999, Proceedings*, pages 187–195, 1999.
[web document](#). 17, 88

Hai Zhuge, T.Y. Cheung, and H.K. Pung. A timed workflow process model. In *Journal of Systems and Software*, volume 55, pages 231–243. Elsevier Science Inc, 2001.
[web document](#). 89

Hai Zhuge, H.K. Pung, and T.Y. Cheung. Timed workflow: Concept, model, and method. In *First International Conference on Web Information Systems Engineering (WISE'00), June 19 - 20, 2000, Hong Kong, China*, volume 1, pages 183–189, 2000.
[web document](#). 89

Curriculum Vitae

Wolfgang L. Gruber
2.9.1966

Home Address:

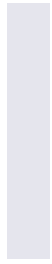
Linsengasse 54
9020 Klagenfurt

Education:

1972 - 1976 Volksschule
1976 - 1980 Hauptschule
1980 - 1984 HTL-Fachschule
Braunau
1987 - 1990 Aufbaulehrgang für Berufstätige
Linz
1990 - 1996 Informatik
Johannes Kepler Universität Linz
1999 - 2003 Doktoratsstudium der Techn. Wissenschaften
Universität Klagenfurt

Work Experience:

1987 - 1989 IVM Engineering
Wien/Linz
1997 - 1999 Siemens Business Services
Augsburg
1999 - 2003 Universität Klagenfurt
Klagenfurt



Electronic Data

This thesis and the described prototype is delivered in electronic form on the enclosed CD-ROM.

Index

Author Index

- Alonso et al. [1997] .. xvii, 10 ff., 86
Becker et al. [2000] 30
Bettini et al. [1998] 92
Bettini et al. [2000] 2, 89, 141
Bettini et al. [2002] .. 2, 85 f., 89, 92, 94, 142
Bonner [1999] 19
Bussler [1998] 2, 87
Casati et al. [1995] 31, 45
Casati et al. [1996a] 19
Casati et al. [1996b] 2, 45
Casati et al. [1999] 88
Chomicki [1995] 36
Clifford and Rao [1987] 92
Curtis et al. [1992] 30
Dadam et al. [2000] 2, 88, 104 f.
Dechter et al. [1991] 93, 142
Dorninger et al. [1990] 105
Eder and Gruber [2002]. 14 – 17, 22, 29 f., 32, 47 f., 51, 86
Eder and Liebhart [1995] 16
Eder and Liebhart [1997] 88
Eder and Panagos [2000] 2, 86, 90 – 97, 104, 109 f., 113, 119, 141
Eder and Pichler [2002] . 25, 93, 106
Eder et al. [1997a] 88
Eder et al. [1997b] ... 14, 16, 18, 20, 137
Eder et al. [1999a] .. 24, 86, 89, 104, 106
Eder et al. [1999b] 30, 86, 89, 92 ff., 96 f., 106, 109 f., 113, 117, 141
Eder et al. [2000] 2, 15, 68, 75, 77, 79, 86, 89, 93, 104, 106, 109 f., 113
Eder et al. [2002] 32, 90
Eder et al. [2003] 35, 141
Eiselt and von Frajer [1977] 105
Ellis and Keddara [2000] 45
Elmasri and Navathe [2000] 123
Flo 85, 93
Galloway [1993] 105
Garlan and Shaw [1993] 124
Georgakopoulos et al. [1995] ... 2, 7, 10, 85
Geppert and Tombros [1997] 90
Glover et al. [1992] 105
Haimowitz et al. [1996] 88
Hofstede et al. [1996] 31
Hollingsworth [1995] xvii, 2, 7 – 10, 85
InC 2, 85, 93
JGraph [2003] 123
Jablonski and Bussler [1996] .. 7, 30
Jablonski [1994] 31
Jasper and Zukunft [1996] 2, 87
Jasper et al. [1996] 92
Joeris and Herzog [1998] 31, 45

- Joeris and Herzog [1999] 13
- Kao and Garcia-Molina [1993a] .. 87
- Kao and Garcia-Molina [1993b] . 87
- Kappel et al. [1995] 19 f., 31
- Kautz and Ladkin [1991] 92
- Kiepuszewski et al. [1999] 15, 48, 71
- Kolmann [2001] 97
- Kradolfer [2000] 30, 32
- Krishnakumar and Sheth [1995] .. 32
- Lawrence [1997] 2, 85
- Leymann and Roller [1994] 2, 85, 93
- Liebhart [1998] 16, 32
- Lin et al. [2002] 15
- Marjanovic and Orlowska [1999a] 23, 88
- Marjanovic and Orlowska [1999b] 2, 30, 48, 88, 94
- Marjanovic [2000] 88, 93
- Marjanovic [2001] 88
- Meyer-Wegener and Böhm [1999] 32
- Modell [1996] 101 f.
- Ninaus [2002] 35, 141
- Panagos and Rabinovich [1996] .. 87
- Panagos and Rabinovich [1997a] . 87
- Panagos and Rabinovich [1997b] . 2, 86 f.
- Perauer and Sorschag [2003] ... 126, 136
- Philipose [1986] 105 f.
- Pozewaunig et al. [1997] xviii, 2, 17, 85 ff., 89, 101, 103, 105 f.
- Pozewaunig [1996] 88, 103
- Reichert and Dadam [1998] 45
- Revesz [1993] 93
- SAP 2
- Sadiq and Orlowska [1999a] 15
- Sadiq and Orlowska [1999b] 30
- Sadiq and Orlowska [2000] . 47 f., 65
- Sadiq et al. [2000] 32
- Schmidt [1996] 2, 85
- Schwalb and Vila [1998] 93
- Suresh and Chaudhuri [1993] ... 141
- Trajcevski et al. [2000] 32
- Ultimus 2
- Warmer and Kleppe [1999] 32
- WfMC [1994a] 8
- WfMC [1994b] 8
- WfMC [1999a] 8, 23, 85, 90
- WfMC [1999b] 2, 15, 32
- WfMC [2000] 7 f.
- WfMC [2001] 7 f.
- WfMC [2002] 7
- Wichert et al. [2001] 19
- Wodtke and Weikum [1997] 18
- Zhao and Stohr [1999] 17, 88
- Zhuge et al. [2000] 89
- Zhuge et al. [2001] 89
- van der Aalst and van Hee [2002] . 7, 13, 17
- van der Aalst et al. [1994] 87
- van der Aalst et al. [1999] .. 46 f., 49
- van der Aalst et al. [2000] 23, 48
- van der Aalst et al. [2002] .. 13 f., 23
- van der Aalst [1998] xvii, 17
- van der Aalst [1999] 45
- van der Aalst [2001] 45

A

activity duration 92
 activity occurrence 16
 ad hoc workflow 10
 administrative workflow 10

B

buffer time 106
 build-time 90

C

collaborative workflow 10
 CPM chart 102

D

deadline 92
 dummy occurrence 97

E

ECA rules 18
 enterprise resource planning 1
 ePERT *see* extended PERT
 ePERT chart 103
 explicit time constraints
 incorporation 109
 extended PERT 103

F

fixed-date constraint 96
 conversion 97

G

Gantt chart 102
 Graphical Workflow Designer .. *see*
 GWfD
GWfD 123
 application layer 124
 architecture 124

child-model window 135
 data source connectivity layer 124
 functionalities 125
 load/save window 127
 main window 126
 model window 130
 presentation layer 124
 specification editing window 128
 specification window 130
 time constraint list 132
 zooming 131

J

Java 123
 JDBC 123
 JGraph 123

L

language
 rule-based 19
 script 18
 lbc *see* lower bound constraint
 lower bound constraint 94

O

occurrence *see* activity occurrence 16
 OCL 32

P

PERT
 extended *see* extended PERT
 PERT chart 101
 Petri net 16
 post-run-time 90
 precedence graph 17
 production workflow 10
 prototype *see* **GWfD**

R

run-time 90

S

schedule 118, 141
 slack time 106
 state charts 17

T

temporal constraints *see* time constraints
 time
 event 92
 ontology 92
 time constraint satisfiability 109
 time constraints 93
 explicit 94
 temporal relationships 94
 implicit 93
 structural *see* implicit time constraints
 time modeling
 process model 90
 timed workflow graph 104
 timed workflow model
 backward calculation 106
 computation 106
 forward calculation 106
 representation 104
 time constraint edge 105

U

ubc *see* upper bound constraint
 UML 32
 upper bound constraint 94

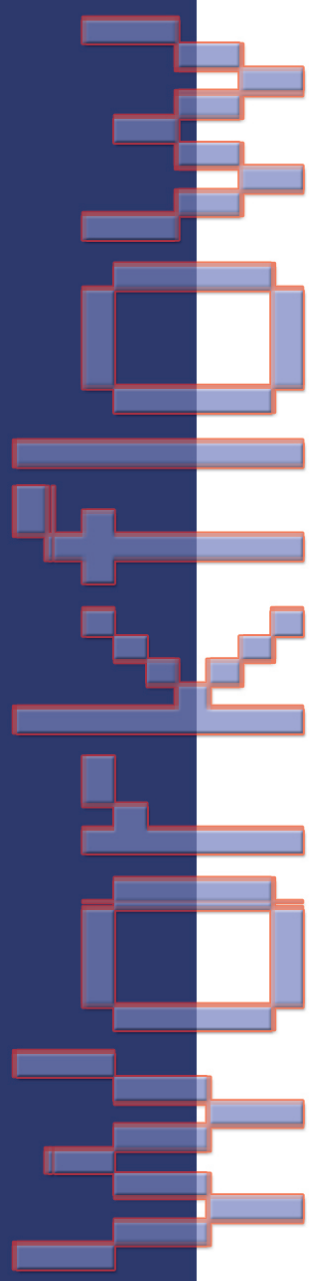
W

well-structured workflow 16
 WfMC 7
 WfMS 7
 workflow
 basic transformation 54

and-join moving over or-join 71
 flatten 55
 join coalescing 65
 join moving over activity occurrence 57
 join moving over seq-join .. 58
 moving alt-join over alt-join 61
 moving alt-join over or-join 64
 moving join over and-join - unfold 68
 moving or-join over alt-join 62
 moving or-join over or-join 59
 moving split before activity occurrence 73
 separating a conditional resp. alternative path 67
 sequence encapsulation 55
 split moving over seq-join . 74
 changes 45
 control perspective 46
 process perspective 46
 resource perspective 46
 system perspective 46
 task perspective 46
 complex transformation 75
 backward unfolding procedure 75
 forward unfold procedure .. 82
 partial backward unfolding procedure 79
 regarding operation ordering 80
 control structures 23
 alternative 24
 conditional 23
 iteration 25
 parallelism 23
 sequence 23
 definition 15
 definition language 18
 equivalence definition 51
 equivalent 51

graph	21
less strictly structured	22
strictly structured	22
history	90
instance type	48
instance types	
equivalent	51
logging	90
metamodel	29
instance level	32
integrity constraints	36
model level	32
organization level	32
specification level	32
stratification	32
timed	99
metamodel	
requirements	30
modeling	13
representation	20
semantic correctness	50
syntactic correctness	50
timed model	98
E-value	98
L-value	98
well structured	16
Workflow Management Coalition ..	7
workflow management systems ...	7
workflow model	
composition structure	26

Systems



Companies and organizations make a great effort to provide better products and better services at a lower cost, of reducing the time to market, of improving and customizing their relationships with customers and, ultimately, of increasing customer satisfaction and the company's profits. These objectives push companies and organizations into continuously improving the business processes that are performed in order to provide services or to produce goods.

Workflow technology has emerged as one of the latest technologies designed to accomplish above demands by modeling, redesign and execution of business processes. Especially in this context, workflow time management is important to timely schedule workflow process execution, to avoid deadline violations, and to improve the workflow turn-around times.

In this thesis, we proposed modeling primitives for expressing time constraints between activities and binding activity executions to certain fixed dates (e.g., first day of the month). Time constraints between activities include lower bound and upper bound constraints. In addition, we present techniques for checking satisfiability of time constraints at process build time. These techniques compute internal activity deadlines in a way that externally assigned deadlines are met and all time constraints are satisfied. Thus the risk of missing an external deadline is recognized early and steps to avoid a time failure can be taken.

Our actual work focuses on:

- (1) providing an advanced workflow metamodel that supports hierarchical composition of complex activities and reuse of activities in several workflow definitions;
- (2) modeling of time and time constraints to capture the available time information;
- (3) developing PERT-net based pro-active time calculations for computing internal deadlines to capture time constraint violations and raise alerts in case of potential future time violations;
- (4) accomplish workflow transformations in order to tackle the problem of unnecessary rejections induced by superfluous time constraint violations; and
- (5) describing our graphical WF-Designer prototype.