Dipl.-Ing. Horst Pichler

Time Management for Workflow Systems
A Probabilistic Approach for Basic and Advanced Control Flow Structures

DISSERTATION

zur Erlangung des akademischen Grades
Doktor der Technischen Wissenschaften

Studium der Angewandten Informatik

Alpen-Adria-Universität Klagenfurt
Fakultät für Wirtschaftswissenschaften und Informatik

1. Begutachter
O. Univ.-Prof. Dipl.-Ing. Dr. Johann Eder
Universität Wien / Institut für Knowledge and Business Engineering

2. Begutachter
O. Univ.-Prof. Dipl.-Ing. Mag. Dr. Gerti Kappel
Technische Universität Wien / Institut für Softwaretechnik und Interaktive Systeme

Dezember/2006
Ehrenwörtliche Erklärung


Dipl.-Ing. Horst Pichler

Klagenfurt, 11. Dezember 2006
Acknowledgements

I am deeply grateful to my mentor Prof. Johann Eder. Without his guidance this thesis would not have been possible. I would also like to thank Prof. Gerti Kappel for peer-reviewing this thesis.

I also wish to thank the people at Groiss-Informatics, namely Herbert Groiss, Michael Dobrovnik, Peter Schelander, and Andriy Vyedyeneyev. They provided very valuable input during the two years of our project and it has been a pleasure to see parts of this thesis being implemented in a real workflow system.

I would like to thank Stefan Vielgut, Mario Lassnig, and Hannes Eichner who aided me, among other things, in the implementation of prototypes and the simulation framework.

I am indebted to many colleagues and fellow assistants for providing stimulating discussions, especially Karl Wiggisser, the author of the style used for this thesis, who additionally had always an answer to my \LaTeX{} questions.

I wish to thank my friends Mike, Michael, Karl, and Jochen (aka "the homies"). Especially our entertaining poker nights provided the sometimes necessary distraction.

I am very grateful to my brother Gerd, my sister Barbara, and my parents. They always cared.

Lastly, and most importantly, I wish to thank Michaela for her patience, love, and understanding. To her I dedicate this thesis.
Abstract

Workflow management systems support the automatization of business processes so that efficiency and productivity are enhanced. An important feature which improves the process quality, is the ability to monitor the compliance to specified constraints. A very frequently applied class of constraints comprises the integration of time and time restrictions, for instance deadlines, into a process. The violation of such a constraint will increase the cost of a process as it often requires expensive escalation-actions to adjust the situation or even entails the payment of penalty fees.

Therefore it is reasonable to integrate mechanisms that allow the prediction and proactive avoidance of time constraint violations. The basic idea is to forecast the execution behavior and task durations of processes by calculating valid execution intervals for each task. This provides the means of monitoring the execution of tasks within their boundaries and to initiate evasive actions if they are prone to violating these thresholds. Although the immense optimization potential of such an idea is rather obvious, existing research approaches still did not receive the attention they deserve in commercial workflow systems. The main reason for that is definitely to be found in uncertainties during process execution. They are, on the one hand, caused by conditional process structures which inhibit exact prediction of upcoming tasks, and, on the other hand, caused by the impossibility of forecasting the exact duration of a task that varies with each execution. Additionally existing research approaches are restricted to very basic workflow control flow elements, which inhibits application in commercial workflow systems.

This thesis describes a probabilistic time management approach, which utilizes empirical knowledge that has been extracted from workflow logs. Forecasts are based on probabilistic timed process graphs, which represents temporal information, like the above-mentioned intervals, in the form of time histograms. These histograms are used to forecast the likelihood of future constraint violations, upcoming activity assignments, and expected remaining execution times. The thesis also shows how to deal with advanced control flow structures with a special focus on arbitrary cycles. Furthermore it describes how to apply proactive time management, such that future deadline violations can be avoided, along with the simulation of several scenarios to demonstrate the benefits of a probabilistic time management approach.
# Contents

1 Introduction ............................................. 1
   1.1 Motivation ............................................. 1
   1.2 Hypothesis and Objectives ......................... 3
   1.3 Structure ............................................. 3

2 Workflow Management Systems ......................... 7
   2.1 Introduction ............................................. 7
      2.1.1 Basic Workflow Concepts ....................... 7
      2.1.2 Historical Overview ............................ 8
      2.1.3 Categorization ................................... 9
      2.1.4 Benefits and Problems ......................... 10
   2.2 Related Work ......................................... 10
      2.2.1 Workflow Systems ................................. 10
      2.2.2 Process Modelling ............................... 11
      2.2.3 Workflow Modelling .............................. 11
   2.3 Workflow Terminology ................................ 12
   2.4 Workflow Management System Architecture .......... 14
      2.4.1 Characteristics of a Workflow Management System 14
      2.4.2 Workflow Reference Model ...................... 15
      2.4.3 Worklists ......................................... 16
   2.5 Process Modelling and Control Flow Representation 16
      2.5.1 Process Perspectives ............................ 17
      2.5.2 The Control Flow Perspective .................. 17
      2.5.3 Graphical Workflow Representation ............. 19
   2.6 Control Flow Elements and Semantics ............... 20
      2.6.1 Basic Structures .................................. 20
      2.6.2 Blocked Structures ............................... 21
      2.6.3 Cyclic Structures ................................ 22
      2.6.4 Advanced Structures ............................ 22
   2.7 Conformance Classes and Sound Processes .......... 23
      2.7.1 WfMC Conformance Classes ...................... 23
      2.7.2 Control Flow Errors .............................. 24
4.4.1 Probabilistic Extended Workflow Graph .................................................. 69
4.4.2 Execution Probabilities ................................................................. 70
4.4.3 Forward and Backward Branching Probabilities ......................... 72
4.4.4 Probabilistic Timed Workflow Graph ........................................... 74
4.5 Calculation of the Probabilistic Timed Graph .................................... 75
4.5.1 Forward Calculation of E-histograms ........................................... 75
4.5.2 Backward Calculation of L-histograms ........................................... 79
4.5.3 Backward Calculation of R-histograms ........................................... 82

5 Interpretation and Application of Time Histograms .......................... 85
5.1 Cumulation and Interpretation of Time Histograms ....................... 85
5.1.1 Ascending Cumulated Time Histograms ........................................ 85
5.1.2 Descending Cumulated Time Histograms ........................................ 87
5.1.3 Slack Time Histograms ................................................................. 90
5.1.4 Histogram Compression ................................................................. 91
5.2 Application at Build Time ............................................................... 93
5.2.1 Slack Times .................................................................................. 94
5.2.2 Satisfiability of a Probabilistic Timed Graph .................................. 94
5.2.3 Identification of Critical Activities and Critical Paths .................. 95
5.3 Application at Process Instantiation ............................................... 95
5.3.1 Adjusting The Deadline ................................................................. 95
5.3.2 Calendar Mapping ...................................................................... 95
5.4 Predictive Time Management at Run Time ....................................... 95
5.4.1 Prediction of Process End and Remaining Duration ...................... 96
5.4.2 Histogram Adjustment ................................................................. 96
5.4.3 Prediction of Deadline Violations ................................................. 97
5.4.4 Prediction of Upcoming Activities .............................................. 98
5.5 Updating the PT-Graph ................................................................. 99
5.5.1 Updating the PT-Graph at Run Time ........................................... 99
5.5.2 Splitting the PT-Graph at Build Time ........................................... 100
5.6 Proactive Time Management at Run Time ...................................... 102
5.6.1 Probabilistic Process Prioritization ............................................. 102
5.6.2 Probabilistic Early Escalation ....................................................... 102

6 Probabilistic Time Management for Cyclic Processes ......................... 103
6.1 Introduction .................................................................................... 103
6.1.1 Problem Statement .................................................................. 103
6.1.2 Existing Methods of Resolution ................................................ 103
6.1.3 Some Problems Remain .............................................................. 104
6.2 Related Work ................................................................................. 104
6.3 Basic Concepts and Definitions ...................................................... 106
6.3.1 Extended Workflow Graph for Cyclic Structures .................................. 107
6.4 Unfolding Arbitrary Cycles ................................................................. 111
   6.4.1 Probabilistic Unfolded Workflow Graph ........................................ 111
   6.4.2 Unfold of Conditional Structures ................................................ 111
   6.4.3 Unfold of Parallel Structures ....................................................... 113
6.5 Calculation of the Probabilistic Timed Graph ......................................... 117
   6.5.1 Forward Calculation ........................................................................ 117
   6.5.2 Backward Calculation ..................................................................... 118
   6.5.3 Histograms and Execution Probability .......................................... 120
6.6 Complexity Considerations ...................................................................... 121
   6.6.1 Complexity Explosion ..................................................................... 121
   6.6.2 Complexity Reduction .................................................................... 122

7 Simulation and Evaluation ......................................................................... 123
   7.1 Objectives .......................................................................................... 123
   7.2 Basic Assumptions ............................................................................. 123
   7.3 Scenario 1 .......................................................................................... 124
      7.3.1 Process Model and Simulation Parameters .................................. 124
      7.3.2 Simulation Results ....................................................................... 125
   7.4 Scenario 2 .......................................................................................... 127
      7.4.1 Process Model and Simulation Parameters .................................. 127
      7.4.2 Simulation Results ....................................................................... 128
   7.5 Evaluation and Discussion of Results ............................................... 129
      7.5.1 Benefits ....................................................................................... 129
      7.5.2 Limitations .................................................................................. 131
      7.5.3 Outlook ....................................................................................... 131

8 Probabilistic Time Management for Advanced Structures ......................... 133
   8.1 Additional Structures in @enterprise ................................................. 133
      8.1.1 Sub-process ................................................................................ 134
      8.1.2 Asynchronous Batch Processing ............................................... 134
      8.1.3 Branch ......................................................................................... 136
      8.1.4 Par-for ......................................................................................... 137
      8.1.5 Or-par ......................................................................................... 137
      8.1.6 Ad-Hoc Workflows ..................................................................... 138
   8.2 Advanced Control Flow Patterns ....................................................... 138
      8.2.1 Multiple Instances ...................................................................... 138
      8.2.2 Cancellation ............................................................................... 139
      8.2.3 Deferred Choice ......................................................................... 139
      8.2.4 Multi Choice ............................................................................... 140
      8.2.5 Synchronization and Multiple Merge ........................................ 141
A.2.11 Hai Zhuge, To-yat Cheung and Hung-Keng Pung ................. 173
A.2.12 Claudio Bettini, X. Sean Wang and Shushil Jajodia .............. 174
A.2.13 Carlo Combi and Giuseppe Pozzi ........................... 174
A.2.14 Jin Hyun Son and Myoung Ho Kim et al. ..................... 175
A.2.15 Wil van der Aalst et al. .................................. 175
A.2.16 Weiping Li and Yushun Fan .............................. 176
A.2.17 Gregório Baggio, Jacques Wainer and Clarence Ellis ........... 176
A.2.18 Ruopeng Lu and Shazia Sadiq et al. ......................... 177
A.2.19 Eder and Pichler et al. ................................ 177

A.3 Related Research Areas ........................................... 178
A.3.1 Workflow Log Extraction and Process Mining .................. 178
A.3.2 Business Process Intelligence and Data Warehouses .......... 179
A.3.3 Simulation-based Scheduling and Forecasts .................. 179
A.3.4 Temporal Concepts in Service-based Applications .......... 180

A.4 Systems .............................................................. 180
A.4.1 Adept ......................................................... 180
A.4.2 Phanta Rheii ............................................... 181
A.4.3 Tibco Staffware ............................................. 181
# List of Figures

2.1 Workflow Terminology [118] ........................................ 13
2.2 Workflow System Characteristics [115] ................. 15
2.3 Graph Variants ................................................... 18
2.4 Basic structures ................................................. 21
2.5 Blocked structures .............................................. 22
2.6 Blocked Cycles .................................................. 22
2.7 Arbitrary Cycles ................................................ 23
2.8 Conformance Classes .......................................... 24
2.9 Examples of Control Flow Errors ........................... 24
2.10 Invalid: Non-blocked Nesting of Control Structures (1) 25
2.11 Invalid: Non-blocked Nesting of Control Structures (2) 25
2.12 Valid: Control Structures of the same Type (1) .......... 26
2.13 Valid: Control Structures of the same Type (2) .......... 26

3.1 Time Management Life Cycle (adapted from [49]) .......... 33
3.2 Time Manager Architecture (adapted from [38]) .......... 34
3.3 Events in a Process (adapted from [20]) .................. 37
3.4 Duration of Activities (adapted from [80]) ............... 38
3.5 Implicit Time Properties and Constraints for Activity B 40
3.6 A Sequential and a Parallel Control Flow Structure ....... 42
3.7 Parallel Execution ............................................... 50

4.1 A Conditional Control Flow Structure ................. 58
4.2 Workflow Graph with Branching Probabilities .......... 64
4.3 Instance Types and Duration Distributions .............. 65
4.4 Time Histogram and Cumulated Time Histogram .......... 66
4.5 Execution Probabilities ...................................... 71
4.6 Execution Probabilities in a Non-blocked Structure .... 71
4.7 Execution Probabilities in a Full-blocked Structure .... 72
4.8 Invalid Execution Probabilities – Unsound Workflow .... 72
4.9 Forward Branching Probabilities ............................ 73
4.10 Backward Branching Probabilities ......................... 74
4.11 Workflow with Conditional and Parallel Structures ..... 75
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2 Process Viewer – Original and Unfolded Graph</td>
<td>148</td>
</tr>
<tr>
<td>9.3 Experiment Runner - Simulation Configuration</td>
<td>150</td>
</tr>
<tr>
<td>9.4 Experiment Runner - Viewing the Simulation-progress</td>
<td>151</td>
</tr>
<tr>
<td>9.5 Experiment Runner - Report</td>
<td>152</td>
</tr>
<tr>
<td>9.6 The Mining Dialog</td>
<td>153</td>
</tr>
<tr>
<td>9.7 The Process Editor</td>
<td>154</td>
</tr>
<tr>
<td>9.8 Time Histograms</td>
<td>155</td>
</tr>
<tr>
<td>9.9 Predictive Features</td>
<td>155</td>
</tr>
<tr>
<td>9.10 The Worklist</td>
<td>156</td>
</tr>
</tbody>
</table>
# List of Tables

3.1 Time Management Approaches .............................................. 32
3.2 Forward Calculation Rules for E-values (adapted from [35]) ................. 43
3.3 Backward Calculation Rules for L-values (adapted from [35]) ................ 44
3.4 Backward Calculation Rules for R-values (adapted from [36]) ................. 44

4.1 Comparison of Structural Support ......................................... 62
4.2 Forward Calculation Rules for E-histograms per Node Type .................. 75
4.3 Backward Calculation Rules for L-histograms per Node Type ................ 80
4.4 Backward Calculation Rules for R-histograms per Node Type ................ 83

5.1 Probabilities of Distances and L-values of Activity D. ...................... 88

6.1 Comparison of Time Management Approaches Structural ..................... 105

7.1 Scenario 1 – Simulation Details (Workload 80%) ................................ 126
7.2 Scenario 1 – Simulation Results for ProbAbort-10 (Workload 80%) .......... 127
7.3 Scenario 2 – Simulation Results (Workload 80%) ................................ 129
7.4 Scenario 2 – Simulation Results for ProbAbort-10 (Workload 80%) .......... 130

10.1 Comparison of Time Management Approaches ................................ 159

A.1 Overview and Comparison of Existing Time Management Approaches .... 167
List of Algorithms

2.1 Forward Topological Sort ................................................. 28
4.1 Execution Probability Calculation ..................................... 71

5.1 Compaction of Ascending Cumulated Time Histograms .............. 92
5.2 Compaction of Descending Cumulated Time Histograms .............. 93
5.3 Cutting Ascending Cumulated Time Histograms ...................... 93
5.4 Cutting Descending Cumulated Time Histograms .................... 94

6.1 Graph Unfold .............................................................. 112
6.2 Expansion of a node – expand(curHitVersion,openNodes,PU,PE) .... 113
6.3 Expansion of an End-par – expandPar(curHitVersion,openNodes,PU,PE) .... 116
Chapter 1

Introduction

Systems for business process automation, like workflow management or enterprise resource planning (ERP) systems, are used to improve processes by automating tasks and getting the right information to the right place for a specific job function [118]. When automatizing processes it is frequently necessary to control the flow of information and work in a timely manner by using time-related restrictions, such as bounded execution durations and deadlines, which are often associated with process activities and sub-processes [33]. Additionally, as automated business processes often span several enterprises, a critical need for companies striving to become more competitive is a high quality of service. Especially expected process execution time and compliance to agreed upon deadlines rank among the most important quality measures [16, 15].

1.1 Motivation

Unexpected delays can lead to time violations which typically increase the execution time and cost of business processes because they require some type of exception handling [85]. Therefore, the comprehensive treatment of time and time constraints is crucial for the design and management of business processes. Process managers need tools that help them predict execution durations, anticipate time problems, proactively avoid time constraint violations and make decisions about the relative process priorities and timing constraints when significant or unexpected delays occur [33]. Even though many commercial products offer sophisticated modelling tools for specifying and analyzing workflow processes, their time management functionality is mostly restricted to the monitoring of constraint violations and to simulation for process re-engineering purposes [20, 35]. In research several attempts have been made to provide solutions for advanced time management features, e.g. [84, 88, 24, 34, 76], which basically address the following problems:

- Workflow modelers need means to represent time-related aspects of activities and temporal inter-relationships between them. Therefore the basic process model, quite frequently represented by a graph-variant, must be augmented with explicit temporal information about time constraints and expected activity durations.
The process structure along with expected task durations and specified time constraints provide the means for calculating time properties for single activities as well as for the whole process. These properties are used for checking the feasibility of the process, determining the expected process duration, evaluating whether specified deadlines can be met at all, and calculating valid execution intervals for activities. The latter restrict the time span for the execution of each activity, such that it must be started and finished within these boundaries in order to meet all precedence and time constraints of the process.

Such forecasts can be utilized during process execution by administrators, participants or – in an automatized fashion – by the system itself, to predict execution times and durations, to plan (schedule) work in advance, or to forecast eventually upcoming deadline violations which allow the initiation of evasive actions to speed up the process in order to meet these deadlines.

Unfortunately almost all existing research approaches suffer from uncertainty about time information and lack of structural support, which hinders their integration into commercial systems.

1. The uncertainty about temporal information has two reasons. First, the duration of a task can vary greatly without any possibility of the workflow system knowing beforehand. Second, in a workflow different paths may be chosen with decisions taking place during the execution. Although some approaches try to address this problem by introducing time intervals for the best and the worst case (e.g. [7, 20, 35, 77]), this is not sufficient when it comes to complex processes with highly distributed durations and many conditional structures.

2. Existing time management techniques support only very basic control flow elements allowing for sequential, parallel and conditional execution, whereas most commercial systems offer a much greater variety of control flow elements with diverse execution semantics.

An additional, very effective motivator, was the funded project 'Probabilistic Time Management in @enterprise'. During the course of this thesis Johann Eder and myself started this research project in cooperation with Groiss-Informatics with the aim of integrating time management in their workflow system @enterprise. The project was partially funded by the FFG – "Österreichische Forschungsförderungsgesellschaft mbH"², which is the central institution for funding and promotion of technology innovations in the area of applied research in Austria.

¹http://www.groiss.com
²http://www.ffg.at
1.2 Hypothesis and Objectives

This thesis aims to prove the following basic hypothesis: the application of probabilistic time management in a workflow system techniques will

- reduce the number of deadline violations,
- reduce the tardiness (amount of lateness) of time-constrained workflows,
- allow forecasts of upcoming activity assignments, and
- allow the prediction of process completion times.

In order to realize such a holistic probabilistic time management approach the following objectives are pursued in this thesis:

**Modelling and Representation of Uncertain Information** One of the main objectives of this thesis is to provide an improved temporal model that captures uncertain information about activity duration as well as the branching behavior of processes, based upon empirical knowledge. Such a representation should comprise all kinds of temporal information, including valid execution intervals, process duration, and so on.

**Calculation Rules for Basic Control Flow Elements** A probabilistic representation of time properties requires applicable calculation rules which incorporate the execution semantics of certain basic control flow elements.

**Provide Solutions for Advanced Control Flow Elements** Another objective is to raise the general acceptance of workflow time management, and probabilistic time management in particular, by providing applicable techniques for more advanced control flow elements, with a special focus on cyclic structures.

**Holistic Time Management Architecture** As most existing techniques are specialized on certain time management issues, it is a necessity to provide the basis for a holistic time management approach that includes several predictive and proactive build and run time features, as well as a description of how to integrate them into the architecture of a workflow management system. For a proof-of-concept of the probabilistic approach according prototypes for modelling and simulation of diverse business process scenarios must be provided.

1.3 Structure

Chapter 2 gives a short overview over workflow management systems and explains basic concepts and terms used throughout the remainder of the thesis. Furthermore it discusses
diverse issues on a high-level which must be considered when modelling business processes and workflows. Then it delves into control flow aspects, as a thorough mastery of execution semantics is essential for the application of workflow time management.

Chapter 3 gives a holistic overview of diverse aspects of workflow time management over all phases in the life-cycle of a business process, based on existing literature. It covers build time issues, like modelling temporal information and constraints, the calculation of a basic temporal model or how to check for tight deadlines, as well as run time issues, like the prediction and proactive avoidance of deadline violations.

Chapter 4 discusses problems of existing approaches caused by uncertainty in detail and presents the basics of probabilistic time management as a possible solution to these problems. It describes how to integrate probabilistic information into the process model by means of branching probabilities and time histograms and defines build-time calculation-rules for valid execution intervals.

Chapter 5 shows how to apply the probabilistic timed graph during build and run time. It explains how to cumulate and interpret time histograms and discusses how existing time management techniques must be adjusted such that they can cope with the information provided in the probabilistic timed graph.

Chapter 6 presents a probabilistic approach that allows dealing with arbitrary cycles in process structures. It proposes an unfold technique which unfolds the process graph and explains necessary adjustments to calculation rules defined in prior chapters.

Chapter 7 presents several process scenarios, provides simulation results, and compares the application of regular to probabilistic proactive and predictive time management techniques in a workflow management system.

Chapter 8 deals with the integration of further enterprise control flow elements as well as other advanced workflow control flow patterns. It describes the execution semantics of all structures and necessary adjustments for the probabilistic model.

Chapter 9 presents prototypical implementations realized during the course of this thesis, the simulation framework and details about the current state of probabilistic time management in enterprise.

Chapter 10 summarizes this thesis, discusses opens issues, and provides a short overview over current and future work.
1.3 Structure

Appendix A provides a thorough overview over the origins of, related work on and systems that apply workflow time management, as well as a brief summary of tangent research areas.
Chapter 2

Workflow Management Systems

This chapter provides an overview over workflow concepts and terminology, architectural issues, and process modelling perspectives needed to formalize and automatize a business process with a focus on the control flow. For workflow time management purposes the control flow is the core concept because different types of calculation operations adhere to different types of control flow elements.

2.1 Introduction

2.1.1 Basic Workflow Concepts

The Workflow Management Coalition (WfMC) published the Workflow Terminology [118] which describes most workflow-related terms and concepts in a very compact way.

- **Process, Business Process** A set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships.

- **Workflow** 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 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, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications.

A very good description can also be found in [51]:

*Workflow management software is a proactive computer system, which manages the flow of work among participants, according to a defined procedure consisting of a number of tasks. It co-ordinates user and system participants, together with the appropriate data resources, which may be accessible directly by the system or off-line, to achieve...*
defined objectives by set deadlines. The co-ordination involves passing tasks from participant to participant in correct sequence, ensuring that all fulfill their required contributions, taking default actions, when necessary.

This description contains every concept that is necessary to explain what a workflow basically is and consists of (see also [13]):

- A number of tasks to be executed in a defined order.
- These tasks are assigned to participants for execution.
- A participant may be human, a machine or an application.
- The execution of a task is based on data, belonging to this task or the whole workflow.

Additionally one can see how a workflow management system fits into this schema.

- It coordinates work by forwarding tasks from participant to participant in a defined order.
- It monitors execution such that defined objectives, constraints or business rules are met.
- It automatically triggers defined or default escalation actions if constraints are violated.

2.1.2 Historical Overview

The support and automatization of tasks has always been one of the core objectives of information technology. Although workflow management systems came not into being until the early nineties, similar concepts and applications already existed then. A system named Scoop was developed in 1977, and another one, named OfficeTalk-D, in 1982. Both aimed at automatization of work and a paperless office, but they did not meet their expectations. The clients and network infrastructures available in the early eighties could not support the suggested concepts [11]. Although the demand and the market for workflow-alike systems already existed, workflow systems could not be realized before the early nineties due to technological reasons [13]. Workflow management, as it is seen today, is actually an amalgamation of several technologies: office automation, business process modelling and re-engineering tools, electronic mail, project planning and support software, document management, and transaction-based applications [115, 13, 22].

In the rather short history of workflow management already four generations are distinguished [11]. The first generation consisted of proprietary applications with integrated hard-coded processes. Several commercial systems were developed, especially in the area of document-management. The second generation featured specification and customization
of processes with specially designed scripting languages. These systems already pursued a generic approach and aimed at supporting processes in arbitrary areas of application. The third generation offered interfaces to communicate with external applications, along with graphical user interfaces to support the design phase, as well as simulation and analysis components. The rise of the internet, followed by intra-net and extra-net, brought systems of the fourth generation, with the goal of connecting and synchronizing distributed inter-organizational business processes based on standardized protocols [116].

2.1.3 Categorization

Workflow management systems can be categorized according to their area of application, e.g. [13] differentiates between coordination systems, administrative systems, and information systems. Another, very frequently cited, workflow categorization schema is based on process types [45, 11]:

- **Administrative Workflows** are applied in bureaucratic and administrative environments, which are well established, are usually based on forms, and participants are already familiar with the process. The implementation of administrative workflows for automatizing already existing manually handled processes is often easily achieved, as forms, participants and task sequences are known.

- **Production Workflows** are used for business critical processes. They have a highly repetitive nature, which implies for instance that even the (exact) duration of single steps will be known in advance. Their structure must be modelled and analyzed very carefully, which includes the specification of constraints. They are critical as the violation of a constraint, e.g. a missed deadline, often results in penalty payments which further increases the costs of the process.

- **Ad-hoc Workflows** have no defined structure. They allow enactment, forwarding, and distribution of tasks and data in a flexible fashion, according to the current needs of and assessments made by participants. For ad-hoc workflows it makes no sense, or it is even impossible, to define a strict process structure that considers every possible situation. **Collaborative workflows** are very similar to ad-hoc workflows, with a special focus on how participants work and communicate with each other. Due to their dynamic nature both types often reduce the workflow system to a mere supporting framework which logs the process, monitors execution, and provides a communication infrastructure [3].

The authors of [27] argue that these categorizations are rather strict and that many workflows as well as workflow systems may belong to several categories. Another categorization is presented in [11], which differentiates by means of technology and architectural issues, for instance e-mail based, client-server, and so on. However, this thesis focuses on production and administrative workflows, which are highly structured and predictable.
2.1.4 Benefits and Problems

The benefits gained by an intelligent use and application of workflow systems are manifold [13, 11, 116] – only to mention some of them:

- **Productivity**: automatized forwarding and assignment of tasks reduces delay and waiting times, and furthermore transport times (of forms) can often be eliminated completely.

- **Quality**: constraining tasks as well as results to certain quality criterions and monitoring their compliance increases the process quality.

- **Trace**: the current state of a process can be determined at any time.

- **Audit**: the system provides the means for auditing arbitrary performance measures of currently running and past processes.

- **Optimization**: processes can be optimized based on reports generated from execution logs.

Naturally, as every advantage usually comes with a disadvantage, several problems can be identified as counterpart to the above-mentioned benefits [13, 57]:

- **Inflexibility**: human participants may become frustrated as the degree of control may be too high, caused by too many constraints and quality criterions.

- **High workload**: waiting, queuing and transport times are reduced or eliminated which implies that there are less breaks which may burn out the participants.

- **Surveillance**: the existence of monitoring and tracing features is very tempting for officers as they provide the means for reducing employees to comparable numbers.

These problems can of course be avoided by introducing workflow systems in an employeesensitive manner, mainly based on information and discussion. Employees will quickly understand that the system – if it is not used as 'Big Brother' – will support them in their daily work and relieve them from time-consuming and annoying tasks. Techniques and ideas presented in his thesis should also be applied in this spirit.

2.2 Related Work

2.2.1 Workflow Systems

Many definitions and concepts, provided in this chapter, are based on findings presented in publications of the Workflow Management Coalition (WFMC). The WFMC is an international non-profit organization founded in 1993. Its members are vendors of commercial
2.2 Related Work

Workflow systems, users, analysts and researchers. They aim at defining standards for architecture, interfaces, exchange protocols and definition languages. Their original mission was to couple workflow systems of different vendors. This resulted in the definition of a common terminology [118], a workflow reference model describing the components and characteristics of a WfMS [115], and a good deal more. For further details please refer to WfMC-publications, to be found at [URL].

2.2.2 Process Modelling

Although workflow modelling and business process modelling are very much alike it is important to note that workflow models have a special focus on technical issues, for instance how to communicate with external applications, whereas business process modelling and re-engineering focuses on introduction, comprehension and optimization of a process. Nevertheless it is sometimes hard to distinguish between workflow and business process modelling particularly as they continuously grow together, e.g. process definition tools which export models to workflow systems or the other way round in re-engineering scenarios. Many models and languages have been proposed for the specification of business and workflow processes. Some of them are based on existing techniques, such as precedence graphs or state charts, whereas others are system specific. The US National Institute for Standards and Technology provided an overview, analysis, and comparative study of several process representations, which was performed during the analysis phase of the Process Specification Language PSL [67]. Categorization attempts and classification schemes can also be found in [79], [126], and [49]. Recent endeavors aim at the development of meta modelling languages, where among the most prominent representatives are: Process specification language PSL, Process interchange format PIF, Unified Modelling Language UML, Business Process Modelling Notation BPMN and the ICAM Definition Languages IDEF [123]. Additionally, in the recent past, a lot of web-services based business process definition languages and interfaces emerged, e.g.: XLANG, WSFL, WSCI or BPEL4WS, which naturally focus on service invocation and inter-process communication [2]. The technology report on CoverPages provides an extensive overview over “Standards for Business Process Modelling, Collaboration, and Choreography”, which deals with all kind of modelling standards and meta models, including workflow and Web Services-related specifications.

2.2.3 Workflow Modelling

Nearly every workflow system provides its own proprietary process definition language. Concept and structure of these languages are based on inherent features and underlying process representation models and tailored to the specific needs of the particular field of application; only to mention a few: MOBILE featuring the Mobile Script Language MSL

\[1\text{http://xml.coverpages.org/bpm.html}\]
FlowMark featuring the FlowMark Definition Language FDL, and @enterprise (formerly known as Panta Rhei) featuring the Workflow Definition Language WDL [48]. Nevertheless several authors tried to categorize models and languages from different points of views. Based on observations and categorizations of [67], [79], [126] and [49] it is possible to distinguish between supporting modelling concepts (e.g. state-charts, role activity diagrams, precedence graphs) and specification techniques (e.g. rule-based, constraint-based, input-process-output). Of course one may come up with further classification schemas, depending on the context chosen. For example [49] additionally differentiates between graphical representation (e.g. as graph) and the text based representation (e.g. programming language style).

Specifically for workflow purposes one has to mention the (meta-)language defined in the WfMC Process Definition Interface [117] and its XML-based successor, the XML Processing Description Language XPDL [119]. Research attempts to define workflow meta models can for instance be found in [54, 91, 28] and in [125], which describes the Workflow Process Definition Language WPDL. The core components of many languages correspond to the components defined in the Basic Meta Model of the WfMC [117]. Additionally many vendors offer graphical tools to aid the specification process, where one representation is favored over others: graph-based representation [64]. One also has to mention the Workflow Patterns Research Group around Wil van der Aalst which provides standard patterns for various process perspectives and standard evaluation techniques in order to categorize workflow systems of different vendors. A similar service is offered by the WfMC – as the exchange aspects between workflow systems get more important they provide a list of vendor conformance based on their reference model interfaces. The aim is to support the exchange between arbitrary internal representations of workflow systems.

2.3 Workflow Terminology

For further explanations it is important to sort out relevant terms as well as to understand coherences between different Workflow concepts and components. Most terms listed below are, partially literal, excerpts from the WfMC’s Workflow Terminology [118]. The relationships between different terms are represented in Fig. 2.1.

- **Process Definition** The representation of a process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system. Many other synonyms are used, e.g. model, model definition, flow diagram or workflow script. Especially in time management-related literature control flow model, control flow definition or control flow specification are frequently used synonyms.

---

2http://www.workflowpatterns.com

3http://www.wfmc.org/standards/conformance.htm
2.3 Workflow Terminology

- **Sub Process** A process that is initiated or called from another process, and which forms part of the overall (initiating) process. Multiple levels of sub processes may be supported.

- **Activity, Node** A description of a piece of work that forms one logical step within a process definition. Frequently used synonyms are node or step. An activity is called *atomic* if it can not be decomposed any further. It is called *complex* or *composite* if it represents is an execution block consisting of multiple atomic or again complex activities [34].

- **Workflow Participant** A resource, specified as performing the work represented by an activity. This resource may be a person, a program, or a machine. A very frequently used synonym is *agent*.

- **Role, Group** A group of workflow participants exhibiting a specific set of attributes, qualifications or skills. In role-based workflows it is typically possible to specify

---

**Figure 2.1: Workflow Terminology [118]**
certain roles for an activity, allowing only participants of this role to perform the activity.

- **Process Instance, Case** The representation of a single enactment of a process. A process instance is created, managed and terminated by a workflow system according to the process definition. Multiple instances may be executed at the same time. Each instance exhibits an internal state, which represents its progress. Very frequently used synonyms are case or workflow case.

- **Activity Instance** The representation of an activity within a process instance. Each activity instance represents a single invocation of an activity and relates to one process instance. A frequently used synonym is step instance.

- **Work Item, Task** The actual work assigned to a workflow participant or a group of participants which always relates to a certain activity instance. A frequently used, quite overloaded, synonym is task as it often stands also for activity as well as activity instance. In this thesis task will solely be used as a synonym for a work item, interpreted as a task to be completed by a certain participant or group of participants.

- **Worklist** A list of work items associated with a given workflow participant. It is the interface between a participant and the workflow system. Note that items in a worklist may stem from different process instances, which themselves may stem from different processes and sometimes even from different workflow engines or systems.

### 2.4 Workflow Management System Architecture

The architecture of a workflow management system is best explained by concepts and models from the WfMC. Their Workflow Reference Model [115] describes the components of a WfMS as well as their characteristics.

#### 2.4.1 Characteristics of a Workflow Management System

When talking about workflows one has in principle to differentiate between the build time and the run time phase, as shown in Fig. 2.2.

#### 2.4.1.1 Build Time (Design Time)

During build time a workflow designer analyzes and defines the process. The result in the formal process definition (also called workflow schema), that describes every aspect of the process such that it can be interpreted and executed by the WfMS. This definition contains among other things the steps of the process, the control flow between these steps, the participants involved, data used, constraints, and so on. A standard for the specification
of (rather simple) process definitions can be found in the Interface-1 of the Workflow Reference Model [117].

2.4.1.2 Run Time (Execution Time)

The workflow process definition acts as a pattern for process execution. Each time a new process is started – this instant is also called process instantiation – the workflow engine creates a new instance of the process, interprets and executes it according to the definition. The engine is the core-component of each workflow system. Apart from driving the process it is responsible for the interaction with workflow participants, it logs every step and it monitors whether instances are still compliant to specified constraints.

2.4.2 Workflow Reference Model

The workflow reference model of the WfMC [115] splits features and capabilities to five components and describes each of them in isolation. These components are called interfaces as their functionality is defined by means of application programming interfaces (APIs). The interfaces are placed around the core workflow enactment service, which may contain several workflow engines:

- **Interface 1 - Process Definition** This interface describes a generic and extensible standard for process definitions, including process structure, participants, constraints, and so on.
2.4.3 Worklists

According to the WfMC [118] worklists are administered by a worklist-handler – a software component that manages the interaction between the user (or group of users) and the worklist maintained by a workflow engine. It enables work items to be passed from the workflow management system to users and notifications of completion or other work status conditions to be passed between the user and the workflow management system. Each time the workflow engine starts a process it generates a new process instance (case). The enactment of an activity for a process instance is called activity instance. Whenever an activity instance is ready for execution according work items will be generated. This thesis concentrates on Workflow Systems with push-oriented work list handlers – the workflow system automatically updates the worklist of participants according to the current state of process instances [63]. In case an activity instance has to be performed by a certain role, or a group of participants, the according work item will pop up in the worklist of every potential performer. If one selects this item for execution, it will be removed from the worklists of other potential performers. When the performer finishes activity execution, she signals this to the engine (via an according worklist-function), and the work item will be removed from his worklist. The engine proceeds according to the control flow and a new work item for each successor activity will be generated in the worklists of the responsible participants.

2.5 Process Modelling and Control Flow Representation

In order to automate business processes with workflow management systems it is necessary to model them first. Divers concepts must be combined, depending on the business
context, to describe the universe of discourse of a business process and to specify the full process with all perspectives necessary for automation.

2.5.1 Process Perspectives

To support modelling, control and execution of processes, a workflow management (or business process automation) system needs information about the process and the organizational and technical environment. This information can be categorized to five perspectives [13]:

1. The control flow perspective or process perspective specifies the execution-order of tasks, using different control structures like sequences, alternatives or parallelism.

2. The organizational perspective specifies the organizational structure and resources.

3. The informational perspective or data perspective deals with control data (or workflow-relevant data, e.g. used for routing-decisions) and production data (or business data, e.g. information objects used accessed by workflow-steps).

4. The functional perspective or task perspective describes the functionality of process steps. A task is a logical unit of work with characteristics like the set of operations to perform a trigger or a due date.

5. The operational perspective describes the elementary operations (one task may consist of several operations). Operations may be executed manually or invoke external applications (e.g. text editor, accounting system, ...).

2.5.2 The Control Flow Perspective

The control flow perspective has a central position in the definition of a business process, since other perspectives, e.g. data flow, rest on it. Additionally it provides an essential insight into a workflow specification's effectiveness [106]. Most workflow models are organized around the the control flow, which provides several advantages [74]: they are very intuitive, as they follow the natural ordering of individual tasks a; they are well-suited for monitoring purposes, as it is easy to determine the current state of execution; and the flow-based model can be augmented with additional information, like the data-flow between tasks. The control flow specifies the execution order of activities, such that tasks can be assigned to participants in this anticipated order. It may not only contain sequences, but also conditional, parallel or cyclic execution of tasks. For workflow time management purposes the control flow is the core concept because different types of calculation operations adhere to different types of control flow elements.
2.5.2.1 Graph-based Control Flow Representation

Numerous representation models are in use, but among all alternatives one class is favored in workflow systems as well as in workflow research papers: a graph-based representation. They are very popular as their clear and simple concepts make them intuitively comprehensible, which is also of advantage when discussing process structures with non-experts or customers [64].

2.5.2.2 Precedence and Directed Graphs

A precedence graph is a directed acyclic graph that contains no directed cycles and no transitive edges [10]. Nodes represent activities and edges indicate the order in which activities are to be accomplished (precedence constraints). The left-hand side of Figure 2.3 shows a precedence graph. The directed edge from node A to node B means that activity A must be finished before B can start. The same applies for A and C. Note that B and C are executed in parallel. Nodes with multiple predecessors, like D, must wait until all their predecessors are finished. Numerous descendant methodologies like PERT and CPM are widely used, especially in project management and product development domains [88]. When it comes to workflow systems mere precedence graphs are not sufficient as the control flow is solely defined on the precedence of activities. This means that as soon as a node finished its execution all connected successors are started. This concept allows parallel execution but for example prohibits conditional execution of tasks, which is an indispensable concept in business processes [12]. Therefore the graph must be augmented with special control nodes, where the type of such a node defines the flow behavior. The center graph of Figure 2.3 shows an example where either activity B or C will be executed after A, depending on a condition c. Precedence graphs with control nodes are often used in time management related publications, for instance in [76, 85, 34]. Since precedence graphs do not allow cycles and transitive edges by definition it is impossible to model structures frequently used in programming languages like loop- or goto-statements. For these situations a plain directed graph augmented with control nodes may be used, which of course eliminates the possibility of reasoning about the graph’s structure using precedence-graph related methods and algorithms. The right most graph in Figure 2.3 represents a process which executes A and C until condition c is satisfied.
2.5 Process Modelling and Control Flow Representation

2.5.2.3 Petri Net Variants

The drawback of simple-graph representations is that they originally do not allow to reason about the execution of a control flow model, which is an inherent feature of Petri nets. A Petri net represents a process as a bipartite directed graph [86]. It consists of place nodes, transition nodes and directed edges (also called arcs) which connect places with transitions. During a Petri net's execution, each place can hold zero or more tokens. A transition fires if a defined number of tokens appear on each of its input places. In this case it consumes the input tokens and places a defined number of output tokens on each of its output places. Petri nets are for instance used to model and examine the behavior of concurrent distributed systems.

Several researchers (e.g. [54, 103]) proposed the use of Petri nets in the workflow domain, as they allow for the modelling of states, events, conditions, synchronization, parallelism, choice, and iteration. The strict mathematical specification and formal semantics of Petri nets allows the application of several analysis techniques, like reasoning about the flow behavior of different control structures in order to find execution problems (for further details see also Section 2.7.3). They allow modelling complex control flow behavior like competitive tasks or the (explicit) cancellation of tasks. Additionally, as Petri nets are state-based, they allow distinction between enabling and executing a task. It is also possible to introduce temporal features (time stamps) into a model by using colored tokens [108]. However they also bear some disadvantages. Even for small models the they tend to get complex and large [103]. It is not possible to specify global constraints that relate multiple states, resources or activities to each other [82]. Some authors even claim that the token-game semantics of classical Petri nets is not suitable for workflow systems at all, and that it has to be extended with reactive features as e.g. with ECA rules [42]. A detailed treatise about advantages and disadvantages of applying Petri nets can be found in [82]. How to map workflow concepts onto Petri nets is for example discussed in [41] or [103]. The latter introduced a new class of Petri nets called Workflow-nets (WF-nets), which were the basis for the analysis of different workflow patterns [106] and for the creation of Yet Another Workflow Language YAWL [105].

2.5.3 Graphical Workflow Representation

In this thesis workflows are represented as directed graphs with control nodes. Activities are displayed as boxes labelled with unique names (usually capital letters). Control nodes are represented as circles, labelled with an abbreviation of the according node type; e.g. OS and OJ for or-split and or-join. If more than one control node of a specific type is needed a number will be attached to the label, e.g. O51 and O72, such that node names are unique in the graph. In the style of Petri nets, start nodes will be displayed with an initial token in the middle of the circle. Where it is necessary to represent the current state of a running process instance the initial token of the start node will be displayed at
the appropriate node(s). End nodes are represented as empty circles. A formal workflow graph definition can be found in Section 2.8.

2.6 Control Flow Elements and Semantics

Although many control flow structures are used and interpreted alike in existing workflow management systems and workflow-related research publications, there was for a long time no real common consensus about the execution semantics of diverse structures. In the mid-nineties the Workflow Management Coalition provided a definition of some basic control flow elements in their terminology [118]. Some years later the group around van der Aalst published some papers, e.g. [106, 114, 109], on basic and advanced control flow elements and structural patterns, which provide a detailed view on the execution semantics.4

2.6.1 Basic Structures

For the following explanations please also refer to Figure 2.4 which shows an example for each structure.

- **Sequence** An activity in a workflow process is enabled after the completion of another activity in the same process [118]. Sequences are represented as a single directed edge between a predecessor and its successor. Frequently used Synonyms are sequential routing and serial routing.

- **Or-split** A point within the workflow where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches [118]. In [114] this structure is also called exclusive choice or xor-split, as exactly one of several branches must be chosen. Another or-split variant, called alternative, is described in [34], which is not distinguished from a regular or-split in this thesis. Note that in workflow systems and research papers the edges after an or-split are often labelled with conditions which must hold in order to execute the adhering successor node (e.g. [49]). Accordingly [119] states that in an or-split an unconditional or otherwise-transition is required if there is a transition with a condition – an undefined result of transition evaluation is not permitted. In this thesis it is assumed that in all cases exactly one successor will be selected. Further frequently used synonyms are switch, decision, and conditional routing. An or-split is represented as control node with the label-prefix OR.

- **Or-join** A point within the workflow where two or more alternative workflow branches re-converge to a single common activity as the next step within the workflow [118]. In [114] this structure is also called simple merge and it is stated that it

4http://www.workflowpatterns.com
will be triggered once any of the incoming transitions are triggered. Frequently used synonyms are xor-join and asynchronous join. An or-join is represented as control node with the label-prefix OJ.

- **And-split** A point in the workflow process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order. Note that where a process includes parallel activities, a process instance may include multiple concurrent threads of execution, which also means that multiple activity instances are executed at the same time [118]. Additionally [119] demands that for an and-split no conditions are permitted; In [114] this structure is also called parallel split. Further frequently used synonyms are parallel routing or fork. An and-split is represented as a control node with the label-prefix AS.

- **And-join** A point in the workflow where two or more parallel executing activities converge into a single common thread of control. Each parallel executing thread is held until the set of all thread transitions to the next activity is completed, at which point the threads converge and the next activity is initiated [118]. In [114] this structure is also called synchronization and they discuss diverse implementation variations. An and-join is represented as a control node with the label-prefix AJ.

### 2.6.2 Blocked Structures

[118] and [114] mention that in some workflow systems basic control flow structures can only be defined in a blocked fashion, which is also a requirement in various workflow research papers (see also Section 2.7.1). A blocked workflow consists of basic building blocks which may be nested but must not overlap. Figure 2.5 shows a workflow that consists of nested blocked structures (sequences, and, or), where each or-split has an according or-join and each and-split has an according and-join. Each block has exactly one entry and exactly one exit-node.
2.6.3 Cyclic Structures

The WfMC defines a cycle as the repetitive execution of one (or more) workflow activity(s) until a condition is met [118]. It is very important to notice that this definition assumes that cycles are blocked. Many programming languages provide different types of blocked cycles, for instance `while`, `for-next` or `repeat-until`. In [114] this type is called structured cycle. They also describe arbitrary cycles which allow interleaving cyclic structures. In some programming languages, e.g. `Basic`, these structures are realized by using the `goto`-statement. Frequently used cycle-synonyms are *iteration* and *loop*. In this thesis cyclic structures are represented as a combination of or-splits, or-joins, and the according forward or backward edges. Figure 2.6 shows examples for blocked cycles, where the upper one nests two post-

\[\text{test loops (repeat-until or do-while), and the lower one contains a pre-test loop (while). Both} \]
\[\text{are modelled by means of nested forward or backward transitions. Figure 2.7 shows examples for arbitrary cycles, both consisting of overlapping forward and backward transitions (goto).} \]

2.6.4 Advanced Structures

Most commercial workflow systems offer additional control flow elements with diverse execution semantics. The research group around van der Aalst examined several systems and came up with a list of advanced control flow patterns [109]. Among these patterns are for instance *multiple choice*, *multiple instantiation* (of activities), and *arbitrary cycles* (already
2.7 Conformance Classes and Sound Processes

2.7 Conformance Classes and Sound Processes

Arbitrary combinations of diverse control flow elements allow the specification of structures which cause problems during process execution. Therefore it is necessary to restrict them somehow.

2.7.1 WfMC Conformance Classes

In their process definition language specifications \cite{117, 119} the Workflow Management Coalition distinguishes three different conformance classes.

- **Non-blocked** There are no structural restrictions for processes of this class. Activities and control nodes may be connected in an arbitrary order (only inhibited by the sound-ness criteria as explained in Section 2.7.3).

- **Loop-blocked** For cycles only a blocked implementation or representation (e.g. complex activity) may be used. Arbitrary cycles are not allowed. The activities and transitions of the process definition must form an acyclic directed graph.

- **Full-blocked** For each join or respectively split there is exactly one corresponding split or respectively join of the same kind. The network structure is restricted to proper (blocked) nesting of splits/joins and loops.

Full-blocked is the most restrictive the three conformance classes (see Figure 2.5 for an example). As already mentioned full-blocked structures can be decomposed to basic building blocks (sequence, and, or, cycle) that may be nested but must not overlap. Workflows adhering to this concept eliminate the possibility of deadlocks, unwanted multiple instances, and livelocks during execution, but they also constrain the degree of modelling freedom enormously. There are workflow systems which support only full-blocked models, e.g. SAP/R3 Workflow, but a lot of systems do not, e.g. Staffware or @enterprise. They use transition-based languages or models which allow concepts alike the goto-statement.
Their process definitions adhere either to the conformance class *non-blocked*, which implies no restrictions on the model, or to the class *loop-blocked*, which restricts only loops to be blocked. See Figure 2.8 for some examples; note that the loop-blocked graph contains a loop which is implemented as complex activity.

### 2.7.2 Control Flow Errors

What seems to be a great advantage for modelers, as non-blocked and loop-blocked are not so restrictive, may be a major threat during Workflow execution. Several run time problems may arise due to control flow errors [93, 104, 66]. Figure 2.9 shows some of them. To represent the current state of execution a Petri-net style token representation has been used, where the initial token of the start node has been moved to the currently active nodes. Please consider that and-splits generate multiple out-tokens out of one in-token and and-joins merge multiple in-tokens into one out-token.

- **Incorrect Usage** Examples for the incorrect usage of control flow structures are an activity with two outgoing edges, a synchronizer with only one incoming edge or unconnected activities.
Deadlock: The example in the figure shows a typical deadlock situation. After the execution of activity A, the token is passed to the succeeding and-join, which has to wait for the second token from activity B, which never arrives because the or-split generated only one token.

Active Termination: The and-split passes a token to each of its successors. The first successor to finish passes a token to the end node which, in some workflow systems, terminates the instance. This results in still active activity instances which are part of already terminated process instances.

Unintended Multiple Execution: The and-split produced two tokens. The or-join passes each received token to its successor C, therefore C may be executed twice (if the instance is not terminated before the second execution of C).

Livelock: Livelocks occur for instance in structures with infinite loops, like the one displayed in the figure. The process instance never reaches an end node.

The examples in Figure 2.10 and Figure 2.11 show that problems are frequently caused by an inappropriate nesting of different control structures in a non-blocked fashion. Arrangements of splits and joins of the same type do not raise problems, as presented in Figures 2.12 and 2.13 (note that or-splits and joins must adhere to xor-semantics).

2.7.3 Sound Workflow Structures

The concept of soundness has been defined in [102] – literally a workflow is sound "... if and only if, for any case, the process terminates properly, i.e., termination is guaranteed, there are
no dangling references, and deadlock or livelock are absent ...". Although soundness has been defined on a Petri-net based model it can analogously be applied on a graph-based model (as demonstrated in [107]). It is also important to notice that on account of the restrictions imposed on full-blocked models these structures are always sound, which renders all of the above mentioned problems impossible.

To detect control flow errors at build time, evaluation tools may be used. In [113] a tool called WoFlan has been presented, which allows to check whether a given workflow structure is sound. For further investigations on how to model correct and sound structures refer also to [93, 104, 66] and [65]. The latter especially deals with the soundness of structures with arbitrary cycles. They show that it is possible to transform each sound structure containing arbitrary cycles to a sound structure with blocked cycles. The problem of this approach is, that this does not include parallel execution of activities. They additionally state that mainly parallel structures in combination with arbitrary cycles result in severe soundness problems, namely jumping in or out of and-blocks often results in deadlocks.

2.8 Formal Model of a Workflow Graph

In this thesis processes are modelled as directed graphs with control nodes. As these graphs are augmented with workflow-specific information they are called workflow graphs.

2.8.1 Definitions

Definition 1 (Workflow Graph): A workflow is represented as a directed graph $G = (V, E)$ that consists of a set of nodes $V$ (vertices) and a set of directed edges $E$. 
Definition 2 (Edge): A (directed) edge $s \rightarrow d \in \mathcal{E}$ addresses an edge between a source node $s$ and a destination node $d$, where $s, d \in \mathcal{V}$. The set of all direct (adjacent) successor nodes of node $s$ is denoted $s.Succ$, such that $s.Succ = \{d \mid s \rightarrow d \in \mathcal{E}\}$. The number of successors $|s.Succ|$ is called the out-degree of this node. Analogously the set of all predecessors of node $d$ is denoted $d.Pred$, such that $d.Pred = \{s \mid s \Rightarrow d \in \mathcal{E}\}$. The number of predecessors $|s.Pred|$ is called the in-degree of this node.

In the core part of this thesis nodes will be restricted to the basic node types start, end, activity, or-split, or-join, and-split, and and-join. Note that any kind of loop can be represented with or-splits, backward or forward edges, and or-joins, therefore it is not necessary to introduce special loop-types.

Definition 3 (Node Types): The set $T$ denotes the following set of basic node types: $\{\text{start, end, activity, or-split, or-join, and-split, and-join}\}$.

Definition 4 (Node): A node $v \in \mathcal{V}$ in the workflow graph represents an activity or control node, where $v.t \in T$ denotes its type.

A workflow graph must additionally adhere to the following basic restrictions: there is exactly one node of type start and exactly one node of type end in the graph. The in-degree of start-nodes is 0 and its out-degree 1, whereas the in-degree of end-nodes is 1 and their out-degree 0. The in and out-degree of activities is 1. The in-degree of split-nodes is 1 and their out-degree greater than one, whereas the in-degree of join-nodes is greater than 1 and their out-degree 1.

In this thesis it assumed that a workflow graph always adheres to the soundness criteria, as defined in [102].

2.8.2 Topological Sort Order

Every acyclic directed graph has a topological order, such that a node is sorted before all nodes it has edges to. This ordering is in general not unique, which poses no further problems on the algorithms presented in this thesis. In the remainder of this paper it is frequently necessary to apply operations on each node in a topological order either from the start-node to the end-node (forward) or vice versa (backward). The forward topological order is determined as described in Algorithm 2.1 (adapted from [23]). The backward topological order is determined analogously, starting with the last node. Note that this algorithms does not work with directed cyclic graphs.
Algorithm 2.1 Forward Topological Sort

Input: Graph $G = (\mathcal{V}, \mathcal{E})$
Output: Sorted sequence of nodes $S$

1: $S := ()$
2: $G' := G$
3: while $\mathcal{V}' \neq \emptyset$ do
4: Select a node $v' \in \mathcal{V}'$, where $v'.\text{Succ} = \emptyset$
5: Append $v'$ to the end of $S$
6: $\mathcal{V}' := \mathcal{V}' \setminus \{v'\}$
7: $\mathcal{E}' := \mathcal{E}' \setminus \{v' \Rightarrow \text{succ}\}$
8: end while
Chapter 3

Workflow Time Management

This chapter provides an overview over the basics of workflow time management, origins and related work, the objectives it tries to achieve, as well as how time management components are integrated in a workflow system. It describes the calculation of a timed graph, based on existing techniques that stem from project planning methods, and shows how to utilize the temporal information, stored in such a graph, at workflow build and run time.

3.1 Introduction

Workflow time management is actually a subsumption of several concepts and techniques with quite different objectives which have one thing in common: *they deal with temporal aspects of workflows, aiming at the optimization of time-constrained processes, such that a constraint-violation-free and timely execution is guaranteed.* Frequently used synonyms are temporal workflow management, timed workflow management, workflow scheduling, temporal workflow modelling, predictive workflow management, and temporal reasoning. Main objectives and features supported by time management approaches are [50, 35, 74, 110]:

- **Temporal Modelling** Build-time feature. Specify time properties, like expected activity durations, and time constraints, like process deadlines. The process must be augmented with additional temporal information as a prerequisite for further time management features.

- **Constraint Satisfaction** Build-time feature. Check the satisfiability of constraints at build time. This aims at determining if process execution without violating time constraints is possible at all, or if for instance deadlines are defined too tightly.

- **Analyze Performance Measures** Build-time feature. Analyze (or simulate) the process to determine time-related performance measures, like the expected average turnaround time.

- **Track and Monitor Execution** Run-time feature. Track and monitor process execution and detect time constraint violations. Most workflow systems offer rudimentary
reactive time management features, as for example to raise an exception when a deadline has been violated.

- **Predictive Time Management** Run-time feature. Make forecasts about the process, like the estimated duration, when the end of the process can be expected, or when future activities will be assigned to participants. This also includes the early detection of time-related problems, such as upcoming constraint violations.

- **Proactive Time Management** Run-time feature. Proactively avoid eventually upcoming constraint violations or overloaded resources. Techniques to achieve this are manifold, like adding resources, change process priorities or rescheduling the rest of the process in order to speed it up.

- **Measure Process Efficiency** Post-run | Build-time feature. Provide empirical tools to reason about diverse properties of processes based on heuristics of extracted data, to for instance answer questions like "How much time did a certain department spend on a certain activity?".

### 3.2 Related Work

This section provides a short overview of origins, existing workflow time management approaches, and systems. For a more thorough survey please refer to Appendix A.

#### 3.2.1 Origins

Workflow time management origins are to be found in diverse areas of research and application, where among the most influential rank project planning methods, scheduling techniques and temporal constraint networks [12, 76, 88]. These techniques provide temporal models and algorithms for the calculation of additional implicit temporal information which can then be further utilized to realize diverse time management functionalities.

- **Temporal Constraint Networks** Many workflow time management publications are concerned with time constraint satisfaction issues, and many of them are based on temporal constraint networks (e.g. [72, 50, 44]). TCNs are a thoroughly researched type of constraint network. They are used to specify temporal constraints between events, for instance a maximum duration between the start of one and the end of another activity. Temporal constraint satisfaction aims at finding (at least) one execution scenario under consideration of all specified constraints. TCN-based approaches primarily aim at checking the satisfiability of constraints at build time.

- **Scheduling Techniques** Regular scheduling-techniques aim at finding an execution plan – a *schedule* with *release dates* – for a given set of steps and machines at process instantiation or during run time [61]. Precedence, temporal, and resource constraints
negatively affect the schedulability of steps [68]. The main scheduling objective is to find a feasible assignment of steps to machines, such that all specified constraints are satisfied. Feasibility is usually based on a quantifiable optimization criteria, like minimizing the number of late jobs, or minimizing the turnaround time [4].

- **Project Management Methods** Many workflow time management approaches are based on project planning methods (PPM) like the critical path method (CPM) or the program evaluation and review technique (PERT). They utilize the process structure and temporal information to calculate valid execution intervals. These intervals are bounded by an earliest point in time an activity can start, determined by precedence constraints, and the latest point in time it must end, such that the deadline will not be violated [88].

Although workflow time management and these techniques share several characteristics, they can not be applied in a straightforward manner. [12, 88, 76] identified several differences and shortcomings which inhibit a straightforward application in a holistic time management approach. E.g., TCNs are primarily applied during build time and scheduling-techniques during run time. PPM-methods are applicable in all phases, but they use a static approach, which does for instance not consider the current workload of the system. Furthermore all techniques are inhibited by structural drawbacks, as they support only a few, very basic, control flow structures, mostly only sequential and conditional execution. Further conceptual differences are discussed in greater detail in Appendix A.1.

### 3.2.2 Workflow Time Management Approaches

Apparently none of the original techniques can be applied for workflow time management without major adaptations. Therefore several authors proposed (partial) solutions, mostly based on one of the above-mentioned original techniques, for some of the problems which inhibit the corresponding technique. Table 3.1 provides an overview of existing research approaches (sorted by publication date of first publication). According to origin and scope of application, some techniques are applicable at build time, others during at run time, and again others in both phases. Further details on thesis-specific attributes of each approach will be listed and compared in the related-work sections of succeeding chapters. For a survey that includes thorough descriptions of each approach please refer to Appendix A.2.

However, all existing techniques specialize on one or the other build or run time feature, and are incapable of supporting all build and run time features listed in Section 3.1. To provide a holistic time management architecture is therefore a rather ambitious request.

### 3.2.3 Systems

Although the field of time management has already received a lot of attention in diverse research areas most currently available workflow products provide little support.
Many commercial products offer sophisticated modelling tools for specifying and analyzing workflow processes, but their time management functionality is still rudimentary and mostly restricted to monitoring of constraint violations and simulation for process re-engineering purposes [20]. In [24, 25, 89] the authors outline some temporal extensions to the ADEPT workflow management system (research prototype), called ADEPTtime. In [27] the authors describe, among other things, the integration of time management features into their workflow system Phanta Rhei (research prototype). Their time management component utilizes the ePERT-approach [88], which forms the basis for the extended probabilistic model presented in this thesis. As for commercial systems only one system with advanced time management capabilities could be identified: Tibco Staffware [100] which offers prediction features based on estimated or empirical data. For details on these systems and tangent areas of interest please refer to the survey in Appendix A.4.

### 3.3 A Workflow Time Management Approach

To provide a consistent model and terminology the descriptions of techniques and features are primarily based on work by Eder et al., which is heavily inspired by project planning methods. They extended the capabilities of PERT (Project Evaluation and Review Technique), such that it can be applied on workflows at build and run time. Explanations are augmented with additional information and citations from related research approaches where necessary.
3.3 A Workflow Time Management Approach

3.3.1 Time Management in the Process Life Cycle

According to [49] several phases in the life cycle of a workflow process can be identified, that are relevant for time management applications. Figure 3.1 shows an adapted point of view, extended with some thesis-specific concepts. A process designer models the process in a system-specific modelling language. From this, a workflow graph can be extracted, which reflects the control flow. Additionally it must be augmented with explicit temporal information, like activity durations and deadlines, which results in the extended workflow graph. Based on this, the timed workflow graph [35] will be calculated. It contains additional implicit temporal constraints for each node, which will be utilized to assess the current temporal status (e.g. likelihood of an upcoming deadline violation) during run time. At this stage the timed graph can already be used to make estimations about the temporal characteristics (expected process duration), to verify the temporal integrity of the process (check satisfiability of temporal constraints), and identify critical paths caused by tight deadlines. On process instantiation the timed graph must be calendar mapped to the current system date and time. Additionally it may be dynamically adjusted to the current system load. During process run time the time management component monitors the execution of each process instance, reacts on constraint violations, predicts upcoming problems (like deadline violation), and proacts in order to avoid these problems. Most workflow systems log events, like start or end of activities, along with time stamps, in the so-called workflow history.

![Figure 3.1: Time Management Life Cycle (adapted from [49])](image-url)
During the \textit{post run time} phase empirical information, for instance activity durations and branching behavior, are extracted from this log. This information can be used to answer questions about the process efficiency as well as to improve the temporal build time model. The utilization of experiences from past process-instance executions increases the forecast accuracy of the system.

### 3.3.2 A Time Management Architecture

To illustrate how to integrate time management concepts into a workflow management system we proposed the following architectural sketch [38]: it consists, as displayed in Figure 3.2, of the (proprietary) \textit{workflow process engine} and the \textit{time manager's build time} and \textit{run time} components. Alternative architecture proposals, aiming at the integration of slightly different time management features, can for example be found in [74] or [21].

![Figure 3.2: Time Manager Architecture (adapted from [38])]
3.3 A Workflow Time Management Approach

3.3.2.1 Time Manager Build Time Components

1. The *parser* loads the workflow process definition, parses it, and generates a workflow graph that corresponds to the specified control flow definition.

2. The *data collector* augments this graph with additional temporal and branching information, like expected activity durations, time constraints, and the expected branching behavior. Some of this information can be extracted from the workflow history. Duration estimations for external activities may also be delivered by third parties which gather and manage quality-relevant attributes of integrated services. Not displayed is an interface that enables business analysts to query the data collector, in order to answer questions about the process efficiency. Alternatively, as proposed in [32], one could also integrate a data warehouse that contains prepared information about process structure and execution heuristics.

3. The resulting *extended graph* is fed into the *timed graph calculator* which generates the *timed graph*.

4. And finally the *timed graph* is stored in the *model database*.

3.3.2.2 Process Engine

1. The (proprietary) *process engine* starts new process instances and controls their execution.

2. During the execution of process instances certain events, like start or termination of process activities, are signaled to the run time component of the time manager. Events and time-stamps are logged to the workflow history.

3. To achieve time management objectives (e.g. proactive avoidance of future deadline violations) the process engine reacts to intervention signals from the time manager.

3.3.2.3 Time Manager Run Time Components

1. When a process is started the process engine sends the corresponding signal to the time manager.

2. The *instance-model mapper* loads the timed graph from the model database and generates a calendar-mapped copy, called *timed instance graph*, for the process instance.

3. The instance-model mapper is also responsible for the synchronization of timed instance graphs with further events it receives from the process engine. Such events are: start or finish a process instance, start or finish an activity instance, and the termination of a process instance. In case it receives a finish or terminate event, it discards the associated timed instance graph.
4. The instance-model mapper is also responsible for actualizing timed instance graphs. If a process instance is much faster or slower than expected it adjusts the models corresponding to the current time. Furthermore, load-based actualization can be applied, as e.g. timed instance graphs tend to get out-of-sync if the work load is unusually higher than predicted.

5. The prediction component (which also includes reactive features) periodically checks the temporal status of all running process instances and raises an exception if time constraints are likely to be violated. Additionally it provides an interface for monitoring the temporal status of each process instance (e.g. likelihood of deadline violations, expected remaining execution time) which may be accessed by users, service requestors, and process administrators.

6. In case a process is late the proactive component jumps into action, it could for instance propose to increase the priority for late instances to speed them up, and send according intervention instructions to the process engine.

3.4 Extending the Workflow Graph

Apart from structural information a process model usually contains additional time-related attributes like the duration of activities or the whole process. This information is, in most existing systems, mainly used for simulation and process re-engineering purposes. Additionally many systems allow the specification of simple process deadlines [117]. The extended workflow graph captures this information. It extends the workflow graph with explicitly specified temporal information. Please note that the extended graph has been specifically introduced for this thesis, as an intermediary representation, which will be utilized to calculate and unfold the timed graph in subsequent chapters.

3.4.1 Temporal Concepts

The representation of temporal information in the extended graph must adhere to a well-defined concept. According to Jasper and Zukunft [58] a linear, discrete, one-side bounded time model is a good choice for representing temporal information and the behavior of processes.

- Linearity enables ordering and therefore comparison of events in time.
- Time is one-sided bounded with a lower bound. It indicates a predefined instant at which a process starts, which is usually 0.
- Time has no upper bound, therefore it is infinite in the future.
- Time is discrete with a universal predefined chronon that is specified in basic time units, like seconds or minutes, according to the granularity required.
Additionally the following basic primitives are distinguished [58, 56]: instant (or point in time), interval and duration. Instants define the occurrence of an (atomic) event in time. An interval is constricted by two instants. And a duration is a time span with a definite or indefinite length, e.g. 6 hours or 10 to 12 month.

Time-related information refers to certain events. Start and end events are used to depict the start and end of a process or certain activities respectively. Figure 3.3 shows an example where each event adheres either to the start or the end of the node v. Note that start depicts the start-event of the first node Start of the process. This analogously applies for end and the end of the process. In the example the duration of control nodes

![Figure 3.3: Events in a Process (adapted from [20])](image)

like Start or End is assumed to be 0 – their start and end-events mark the same point in time. Note that in two successive events, like the end of B and the start of C, must not necessarily occur immediately one after another other. There may be a delay, also called transition time [111, 122], between them.

### 3.4.2 Explicit Time Properties and Constraints

Explicit time properties and constraints are explicitly specified by workflow designers during workflow build time. They are derived from expert estimations, past experiences, organizational rules, laws, and commitments.

#### 3.4.2.1 Duration

The expected duration of a node or activity can be represented as (estimated, expected) average value. As proposed in time management and scheduling-related papers an activity duration may additionally be split up into several parts like queue time, preparation time, wait (or idle) time and execution (or work) time; see e.g. [80, 96]. The example in Figure 3.4 shows a scenario with queuing and interrupted execution of activities in the workflow sequence presented above: activity A is ready for execution and is queued in the worklist until the responsible participant (or agent) decides to take and execute it. She works on it for a while and finishes it after a short pause. The workflow system forwards it to the next participant responsible for the execution of the succeeding activity B, who executes it
without delay. The participant responsible for C starts to execute it after a short time and finishes the workflow.

![Figure 3.4: Duration of Activities (adapted from [80])](image)

For the time being a scalar representation is used, where \( v.d \) depicts the duration of a node \( v \in \mathcal{V} \). Furthermore it is assumed that the duration of an activity, bounded by a start and an end event, subsumes all phases – represented as \( A.d, B.d \) and \( C.d \) in Figure 3.4. Additionally it is assumed that the start-event of a node occurs immediately after the end-event of its predecessor. Such a delay of 0 time units simplifies the calculation of implicit time properties enormously. Additionally control nodes are assumed to have a duration of 0, therefore one can state that \( start_{Start} = end_{Start} = start_A, end_A = start_B \) and \( end_B = start_{End} = end_{End} \). Note that a delay (or transition time) between two nodes can be modelled easily by introducing a dummy activity between them, where the delay is represented by the duration of the dummy.

### 3.4.2.2 Explicit Time Constraints

Examples of time constraints are: a high-priority claim must be processed within one day, an invitation to the board meeting taking place every six months has to be sent at least two months before the meeting, or vacant positions must be announced on the first Wednesday of a month. Considering these examples one can see that explicit constraints are either temporal relations between events or bindings of events to certain calendar dates. Most workflow systems offer the capability to specify process deadline constraints and reactive time-management features – usually they raise an exception when detecting a deadline violation during run time.

- **Deadlines** A deadline according to [117] is a point in time where something has to be completed, thus it usually refers to the end of the process. As the absolute date of a point in time, marked by a deadline, is unknown at build time, it is often specified as *maximum process duration* [35], also called *process duration constraint* [76]. It depicts a relative time constraint between the start and the end of the process. At run time such a *relative deadline* will be mapped to an *absolute deadline* according to a real calendar [35]. Since a deadline is externally assigned it is also called *external deadline* [49].
3.4 Extending the Workflow Graph

- **Maximum Durations** It limits the *maximum duration* [20] of an activity and is therefore also called *limited duration constraint* [76]. In some approaches this type of constraint is inherently captured by specifying a [min,max]-interval for the allowed or expected domain of an activity duration, as for instance proposed in [76, 50].

- **Advanced Time Constraints** An *upper bound constraint* specifies that the temporal distance between two events must be smaller than or equal to a given maximum time value. Whereas a *lower bound constraint* specifies that the temporal distance between two events must be greater than or equal to a given minimum value, e.g. to ensure waiting time between two non-adjacent tasks [35]. As these constraints are specified between two events or activities, they are also called *interdependent constraints* [76], *relative constraints* [78] or *inter-task constraints* [20]. A *(periodic) fixed-date constraint* expresses that an event can only occur on a certain date or on a defined list of certain dates, e.g. activity A must start at 8 a.m. on a Monday [35]. In some approaches, especially those based on temporal constraint networks, the whole control flow is represented by means of *temporal relation constraints* or *schedule-task constraints*, as e.g. in [20, 50]. They are used to specify and constrain an allowed order of tasks (activities) and events. For example: the start of activity B must not occur before the end of activity A – this simply expresses sequential execution of A and B. They allow modelling of complex temporal relationships and constraints, e.g. [50] differentiates between 13 temporal relations, for instance *overlaps, during, meets* or *finished by* (see also related work in Appendix A.2).

Please note that modelling of advanced constraints and constraint satisfaction (based on a probabilistic model) is subject of ongoing work and therefore outside the scope of this thesis, which captures only regular deadlines. Advanced time constraints are mentioned for the sake of completeness only. For an application that already utilizes fixed-date constraints in an adapted probabilistic temporal model please refer to our publications [8, 9].

3.4.3 Definition of the Extended Workflow Graph

The extended workflow graph augments the basic workflow graph (defined in Section 2.8) with explicit time properties and constraints.

**Definition 5 (Extended Workflow Graph):** An extended workflow graph $G_E = (V, E, \delta)$ consists of a set of extended nodes $V$ (vertices), a set of directed edges $E$\(^1\) and a process deadline $\delta$.

**Definition 6 (Extended Node):** A node $v.t \in V$ of an extended workflow graph represents an activity or control node, where $v.t \in T$ depicts its type and $v.d$ its duration.

\(^1\)Already defined in Section 2.8.
For the time being throughout this chapter it is assumed that only acyclic blocked or non-blocked workflow graphs, which consist of unconditional structures, where \( T = \{ \text{start, end, activity, and-split, and-join} \} \), are allowed.

### 3.5 Calculating the Timed Graph

Based on the extended graph the \textit{timed workflow graph} [34] will be calculated. It contains additional \textit{implicit time constraints} for each node. The concepts for the calculation of implicit time constraints and properties presented below are rooted in project planning methods like the Critical Path Method (CPM) or the Program Evaluation an Review Technique (PERT). They have been extended with workflow-specific features by e.g. [34, 76]. For the time being the following explanations focus solely on unconditional process structures. Problems of and how to deal with conditional structures and the problems that arise with their existence are discussed in Chapter 4.

#### 3.5.1 Implicit Time Constraints and Properties

Some time constraints follow implicitly from the process structure, activity durations, and explicit time constraints. Thus they are called \textit{structural} or \textit{implicit} [35]. Temporal implicit dependencies arise from:

1. **Precedence Constraints** An activity can only start when its predecessor activities are finished.

2. **Time Constraints** Due to explicitly specified time constraints, like deadlines, activities must end until a certain point in time to ensure an execution without constraint violation.

![Figure 3.5: Implicit Time Properties and Constraints for Activity B](image)

Figure 3.5 shows a workflow that consists of three activities executed in sequence. The duration estimations in basic time units are \( A.d = 3 \), \( B.d = 4 \) and \( C.d = 5 \); the durations of
the control nodes are \( \text{Start} . d = \text{End} . d = 0 \). Additionally a process deadline of \( \delta = 15 \) has been specified. According to ePERT it is possible to utilize the process structure and time information to calculate the earliest point in time an activity can start and the latest point in time it must end, such that the deadline will not be violated [88] (see also Appendix A). The timeline in figure 3.5 shows implicit time constraints and properties, and the valid execution interval for activity \( B \). According to [58] a relative time scale is used, where 0 denotes the start time of the workflow. All other points in time are declared or calculated relative to this start time.

**Earliest Possible Start (EPS)** The implicit time constraint called *earliest possible start time* is determined by the sum of predecessor durations. It depicts the earliest point in time an activity can start. In case of the current example it is \( B . \text{eps} = 3 \).

**Earliest Possible End (EPE)** Accordingly the *earliest possible end time* depicts the earliest possible point in time an activity can end. It is determined by adding the duration of an activity to its earliest possible start time: \( B . \text{eps} = A . \text{eps} + B . d = 7 \).

**Latest Allowed End (LAE)** By taking the deadline into account and reversing the point of view, now starting from the end of the workflow, it is possible to determine an implicit time constraint for each node: the *latest allowed end time*. It is determined by subtracting the sum of successor durations from a given process deadline. The LAE depicts the point in time where the activity must be finished, otherwise a deadline violation is very likely to occur in the future. In case of the running example it is \( B . \text{lae} = \delta - 5 = 10 \). If \( B \) ends until 10 it is still possible to reach the process deadline of 15, as the rest of the process will presumably last about 5 time units.

**Latest Allowed Start (LAS)** Accordingly the *latest allowed start time (LAS)* for \( B \) is determined by subtracting the duration of the activity from its LAE: \( B . \text{las} = B . \text{lae} - B . d = 6 \).

**Valid Execution Interval** The execution interval in which \( B \) can and must be executed is \([B . \text{eps}, B . \text{lae}]\). Activity \( B \) can not start earlier than \( B . \text{eps} \) and must not end later than \( B . \text{lae} \).

**Slack Time (SS/SE)** The *slack time* is spare or buffer time. It can be consumed without violating any time constraints [34]. It arises if the duration of an activity is less than the time-span defined by the valid execution interval. E.g. for \( B \): in the ideal case \( B \) starts at 3 and ends at 7, thus a slack of 3 is generated, because \( B \) is allowed to last until 10. Slack, measured at the end of the activity, is calculated as \( B . \text{se} = B . \text{lae} - B . \text{epe} = 3 \). The slack at the start of \( B \) is therefore \( B . \text{ss} = B . \text{las} - B . \text{eps} = 3 \). Note, that in a temporal model with scalar time values \( B . \text{ss} \) will always be equal to \( B . \text{se} \):

\[
\begin{align*}
\text{se} & = \text{ss} \\
\text{lae} - \text{epe} & = \text{las} - \text{eps}
\end{align*}
\]
Remaining Time (RS/RE) The time property remaining time [120, 36] depicts the expected remaining duration of the process, measured from the according node. For the end of B the remaining time is \( B.re = 3 \), which is equal to \( C.d + End.d \). And for the start of B the remaining time is \( B.rs = 7 \), which is equal to \( B.d + B.re \). The remaining time can for instance be used to inform service-requestors (workflow initiators, customers) about the expected rest duration of a process.

3.5.2 Definition of the Timed Workflow Graph

The timed workflow graph augments the extended workflow graph (defined in Section 3.4.3) with implicit time properties and constraints.

Definition 7 (Timed Workflow Graph): A timed workflow graph \( G_t = (V, E, \delta) \) consists of a set of timed nodes \( V \) (vertices), a set of directed edges \( E \), and a process deadline \( \delta \).

Definition 8 (Timed Node): A node \( v.t \in V \) in a timed workflow graph represents an activity or control node, where \( v.t \in T \) depicts its type, \( v.d \) its duration, \( v.eps \) its earliest possible start time, \( v.epe \) its earliest possible end time, \( v.las \) its latest allowed start time, and \( v.lae \) its latest allowed end time. The properties \( v.ss \) and \( v.rs \) denote slack and remaining time at the start of \( v \), and \( v.se \) and \( v.re \) slack and remaining time at the end of \( v \).

3.5.3 Calculation Rules for Unconditional Structures

All project planning-related techniques have one thing in common: implicit time constraints are calculated in two phases: a forward calculation phase for earliest-possible values, and a backward calculation phase for latest-allowed values. The following outlines and discusses these phases, based on two graphs which contain only sequences and parallel structures: \( SEQ \) and \( PAR \), presented in Figure 3.6. How to deal with conditional structures is presented in the next Chapter.

- **SEQ**
  - Node A with start time 3
  - Node B with start time 4
  - Node C with start time 5
  - Deadline: 12

- **PAR**
  - Node A with start time 2
  - Node B with start time 4
  - Node C with start time 5
  - Deadline: 12

**Figure 3.6: A Sequential and a Parallel Control Flow Structure**
3.5 Calculating the Timed Graph

3.5.3.1 Forward Calculation of E-values

To determine EPS and EPE-values of all nodes in a graph it must be traversed in a forward topological order, e.g. SEQ: \(\text{SEQ}: \langle \text{Start}, \ A, \ B, \ C, \ \text{End} \rangle\) and PAR: \(\text{PAR}: \langle \text{Start}, \ A, \ AS, \ B1, \ B2, \ AJ, \ C, \ \text{End} \rangle\). The calculation rules specified in table 3.2 have to be applied on each node \(n\), where the operation to apply is specified by the node type.

<table>
<thead>
<tr>
<th>(n).type</th>
<th>(n.\text{eps} = )</th>
<th>(n.\text{epe} = )</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>0</td>
<td>(n.\text{eps} + n.d)</td>
</tr>
<tr>
<td>activity, and-split, end</td>
<td>(p.\text{epe}, \exists p \in n.\text{Pred})</td>
<td>(n.\text{eps} + n.d)</td>
</tr>
<tr>
<td>and-join</td>
<td>(\max {p.\text{epe} \mid p \in n.\text{Pred}})</td>
<td>(n.\text{eps} + n.d)</td>
</tr>
</tbody>
</table>

Table 3.2: Forward Calculation Rules for E-values (adapted from [35])

- **Start** The forward calculation starts with the initialization of the start node’s EPS-value: \(\text{Start.}\text{eps} = 0\). The EPE-value \(\text{Start.}\text{epe} = \text{Start.}\text{eps} + \text{Start.}\text{d}\) is calculated by adding the duration of the node to its EPS-value. Note that EPE-calculation is equal for every node type.

- **Activity, And-split, End** Assuming that there is no waiting time between nodes the EPS-value of a node with one predecessor is always equal to the EPE-value of its predecessor.

- **And-join** As an and-join synchronizes execution by waiting until all predecessors are finished, its EPE-value is determined by the maximum EPE-value of all predecessor nodes.

According to this, still assuming that the duration of control nodes is 0, the EPS and EPE-values for SEQ and PAR are:

- **SEQ**: \(\text{SEQ}: \text{Start.}\text{eps} = \text{Start.}\text{epe} = A.\text{eps} = 0, A.\text{epe} = B.\text{eps} = 3, B.\text{epe} = C.\text{eps} = 7, C.\text{epe} = \text{End.}\text{eps} = \text{End.}\text{epe} = 12\)

- **PAR**: \(\text{PAR}: \text{Start.}\text{eps} = \text{Start.}\text{epe} = A.\text{eps} = 0, A.\text{epe} = AS.\text{eps} = AS.\text{epe} = B1.\text{eps} = B2.\text{eps} = 3, B1.\text{epe} = 7, B2.\text{epe} = 4, AJ.\text{eps} = \max(B1.\text{epe}, B2.\text{epe}) = 7, AJ.\text{epe} = AJ.\text{epe} = C.\text{eps} = 7, C.\text{epe} = \text{End.}\text{eps} = \text{End.}\text{epe} = 12\).

3.5.3.2 Backward Calculation of L-values

To determine LAE and LAS-values of nodes in the graph must be traversed in a backward topological order. Following this order the calculation rules specified in Table 3.3 have to be applied on each node, where the operation to apply is specified by the node type. The calculation starts at the last node by initializing its latest allowed end time with the given deadline: \(\text{End.}\text{lae} = \delta = 12\). The LAS-value of each node is determined
by subtracting the duration from its LAE-value: \( \text{End.las} = \text{End.las} - \text{End.d} \). In contrast to the forward calculation now split-nodes need a special treatment, as values from multiple successor nodes must be merged. At the and-join a minimum function must be used (considering only the path with the longest execution duration into account). Still assuming that the duration of control nodes is 0, the EPS and EPE-values for SEQ and PAR are:

**SEQ:**
\[
\text{SEQ: End.lae} = \text{End.las} = \text{C.lae} = \delta = 12, \quad \text{C.las} = \text{B.lae} = 7, \quad \text{B.las} = \text{A.lae} = 3, \quad \text{A.las} = \text{Start.lae} = \text{Start.las} = 0.
\]

**PAR:**
\[
\text{PAR: End.lae} = \text{End.las} = \text{C.lae} = \delta = 12, \quad \text{C.las} = \text{AJ.lae} = \text{AJ.las} = \text{B1.lae} = \text{B2.lae} = 7, \quad \text{B1.las} = 3, \quad \text{B2.las} = 6, \quad \text{AS.lae} = \min(\text{B1.las}, \text{B2.las}) = 3, \quad \text{AS.las} = \text{A.lae} = 6, \quad \text{A.las} = \text{Start.lae} = \text{Start.las} = 0.
\]

### 3.5.3.3 Backward Calculation of R-values

Analogously, to determine the remaining times, the graph must be traversed in a backward topological order. The calculation rules specified in Table 3.4 have to be applied on each node, where the operation to apply is specified by the node type.

**Table 3.4:** Backward Calculation Rules for R-values (adapted from [36])

<table>
<thead>
<tr>
<th>n.type</th>
<th>n.re =</th>
<th>n.rs =</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>0</td>
<td>n.re + n.d</td>
</tr>
<tr>
<td>activity, and-join, start</td>
<td>s.rs, ( \exists s \in n.Succ )</td>
<td>n.re + n.d</td>
</tr>
<tr>
<td>and-split</td>
<td>( \max{s.rs</td>
<td>s \in n.Succ} )</td>
</tr>
</tbody>
</table>

**SEQ:**
\[
\text{SEQ: End.re} = \text{End.rs} = \text{C.re} = 0, \quad \text{C.rs} = \text{B.re} = 5, \quad \text{B.rs} = \text{A.re} = 9, \quad \text{A.rs} = \text{Start.re} = \text{Start.rs} = 12.
\]

**PAR:**
\[
\text{PAR: End.re} = \text{End.rs} = \text{C.re} = 0, \quad \text{C.re} = \text{AJ.re} = \text{AJ.rs} = \text{B1.re} = \text{B2.re} = 5, \quad \text{B1.rs} = 9, \quad \text{B2.rs} = 6, \quad \text{AS.re} = \max(\text{B1.rs}, \text{B2.rs}) = 9, \quad \text{AS.re} = \text{AS.rs} = \text{A.re} = 9, \quad \text{A.rs} = \text{Start.re} = \text{Start.rs} = 12.
\]

### 3.5.3.4 Slack Time and the Critical Path

**Slack Generation** Slack time is, as already mentioned, the amount of spare time, that can be consumed during run time without violating any (implicit) time constraints. During build time slack is predictable for two cases:
1. Parallel structures produce slack. An and-join waits until all branches, nested in the parallel structure, are finished. Therefore, for every node on a "shorter" (in a temporal sense) parallel branch slack time is automatically available. In the PAR-process the node B1 has no slack, as it resides on the longest path, whereas B2 has a slack of \( B2.ss = B2.las - B2.eps = 3 \) (or \( B2.se = B2.lae - B2.epe = 3 \)). This means that B2 may consume up to 3 additional time units without endangering the deadline.

2. A relaxed deadline that is greater than the expected overall process duration also produces slack. E.g., relaxing the deadline of the SEQ-process to 15 produces a slack of 3 for every node. Of course this does not mean that every node may consume 3 additional time units; if for instance A consumes 2 additional time units during run time there is only 1 time unit of slack left for its successors.

**Slack Distribution** As indicated above, slack time, although defined for each single activity, depicts the buffer available for the whole process. The strategy that allows one activity to consume the whole buffer is called total slack \([61]\) – it is obviously rather unfair as it eliminates buffer time for all successors. When no buffer time is left, even slight delays will result in deadline violations. To avoid this several authors, e.g. \([61, 85]\), proposed to distribute slack among remaining (succeeding) activities.

- **Equal Slack** This method distributes the total slack equally among all remaining activity instances. Applied on SEQ, with a deadline \( \delta = 15 \), this results in \( B.st' = \frac{1}{2} \times 3 \) and \( C.st' = \frac{1}{2} \times 3 \).

- **Proportional Slack** This method distributes the total slack among all remaining activity instances in proportion to their durations. Applied on SEQ, with a deadline \( \delta = 15 \), this results in \( B.st' = \frac{4}{5} \times 3 \) and \( C.st' = \frac{2}{5} \times 3 \).

Although proposed for run time application, these techniques are also perfectly suited to be applied at build time. They can be used to reduce the latest allowed end time of the nodes accordingly, e.g. \( B.lae' = B.lae - (B.st - B.st') \).

**Critical Path** Too tight deadlines result in negative slack, which indicates that a deadline violation is very likely. Therefore the calculation of slack time provides the means for finding bottlenecks in the model and of verifying the satisfiability of constraints (see also Section 3.5.4). In this thesis the critical path depicts the path – a sequence of nodes between the start and the end node of the process – with the longest duration, and activities on this path are called critical activities \([18]\). Critical activities have usually low or even no slack time, which means that only very limited or no buffer time is available, and even slight delays will result in a deadline violations. To avoid deadline violations the designer and the process manager should keep an eye on these activities. In order to do this it is necessary to identify the critical path along with its critical activities as for example described in \([18, 98]\).
3.5.4 Satisfiability of Time Constraints

Checking the satisfiability of time constraints means validating if a constraint-free execution of the process is possible at all. It has been stated that a workflow model may be correct in terms of control flow but still have inconsistent temporal constraints [76]. Additionally a temporal constraint is consistent with a given workflow model, if and only if it can be satisfied based on the syntax of the workflow model and expected durations of workflow tasks. For processes with simple deadlines this can be easily checked by examining slack times or the critical path as described above. For models that allow the specification of advanced constraints between events or activities it is more complex difficult. Several techniques can be found in existing time management literature to solve this build time problem. Among the most popular are approaches based on temporal constraint networks, which basically try to find at least one possible execution scenario – a schedule – under consideration of all specified constraints (see also Appendix A.1). Furthermore several techniques exist that have been inspired by project planning methods. For instance [35] describes how to integrate upper bound, lower bound and fixed-date constraints into a temporal workflow model, and how to check their satisfiability. Basically they propose to adjust E and L-values according to the specified constraints and to check if there are any nodes with slack less than 0. Please note that checking the satisfiability of advanced time constraints is subject of ongoing work and therefore outside the scope of this thesis.

3.6 Time Management at Run Time

The time manager’s run time component is responsible for the synchronization between running process instances and the timed graph, which forms the basis for predictive and proactive time management features.

3.6.1 Process Instantiation

A copy of the timed graph must be generated for each new process instance. This copy is called timed instance graph or instance graph, which is used to map relative build time dates to absolute run time dates.

3.6.1.1 Temporal Information at Run Time

In subsequent sections the following terms, each indicating a run time-specific instant in time, will be used: now depicts the current date and time, e.g. now = 8:30 a.m., July 1, 2006. As already mentioned, e_v depicts the actual date and time of an event e = start | end adhering to a node v ∈ V. Consider the following example based on the SEQ-process displayed in Figure 3.6.
3.6 Time Management at Run Time

- Assume that the process started today at 8 a.m., immediately followed by the execution of A, which lasted 2 hours and 50 minutes, immediately followed by the start of B that is still executed. Therefore the actual times of events are (dates omitted):
  \[ \text{start}_A = \text{start}_B = 8 \text{ a.m.}, \text{ and } \text{end}_A = \text{start}_B = 10:50 \text{ a.m.} \]

The actual date and time of the start of a process instance is therefore always depicted by \( \text{start}_v \). Furthermore \( v.d_{\text{act}} = \text{end}_v - \text{start}_v \) represents the actual duration of an instance of the node \( v \in V \).

- The actual duration of activity A is \( A.d_{\text{act}} = \text{end}_A - \text{start}_A = 2:50 \).

3.6.1.2 Calendar Mapping

Every relative time point in the timed instance graph must be replaced with real dates – (absolute time points, calendar dates). The procedure is called calendar mapping: add \( \text{start}_v \) to every implicit time property or constraint in the graph. In this thesis a calendar mapped implicit time constraint is denoted by a subscripted \( \text{cal} \), e.g. \( A.d_{\text{act}}^{\text{cal}} \) is the calendar mapped date and time of \( A.d_{\text{act}} \).

- According to the timed graph implicit constraints for B are \( B.d = 3, \ B.epe = 7, \ B.las = 6 \) and \( B.lae = 10 \). Implicit time constraints are adjusted by adding \( \text{start}_v \) = 8 a.m. This results in \( B.d_{\text{act}}^{\text{cal}} = 11 \text{ a.m.}, \ B.epe_{\text{cal}} = 3 \text{ p.m.}, \ B.las_{\text{cal}} = 2 \text{ p.m.} \) and \( B.lae_{\text{cal}} = 6 \text{ p.m.} \) for activity B.

3.6.1.3 Mapping Deadlines

As time constraints must be monitored during run time it is also necessary to map them to a real calendar. Deadlines can be mapped immediately at process instantiation.

- The overall process deadline is defined as \( \delta = 15 \) (hours). As the process started at \( \text{start}_v = 8 \text{ a.m.} \) this must be mapped to \( \delta_{\text{cal}} = 11 \text{ p.m.} \). This means that the process must be finished by 11 p.m.

In the case that deadlines are changed at process instantiation or during process execution the timed instance graph has to be adjusted accordingly.

3.6.2 Reactive Time Management

Reactive time management can be implemented as a rather simple mechanism that continuously checks if time constraints have been violated.

- Assume that the process still runs and the current time is already now = 11:30 p.m. The time manager detects a deadline violation as now is greater than the calendar mapped deadline 11 p.m. and raises a predefined exception.
Most regular escalation reactions [88, 31], like retry or rollback with forward re-execution, are not suited of handling time constraint violations. They do not eliminate the problem, quite the contrary applies, they make it even worse. Ignore is obviously not suitable for deadline-oriented processes, especially when constraint violations are punished with penalties. To cancel the process may not always be an option at all, and even if it were, resources would have been wasted without achieving a result. To warn or notify the process administrator is an obligatory reaction anyway, which leaves extending the deadline in agreement with the customer or service requestor as a reasonable last alternative [34]. But note that such a deadline extension may also come with a price label attached to it, in the form of a penalty payment.

Workflow systems usually feature escalation reactions such as send email, start escalation process (to be implemented by the workflow designer), and start procedure (to be implemented by a programmer) – which simply means that they do not offer standardized escalation-handlers but leave this to the workflow designers. Additionally [84] distinguishes between escalations with local and global scope. Local escalations affect only the current activity (e.g. restart or replace it), whereas global escalations affect the whole process (e.g. cancel it). However, in all cases an escalation increases the cost of the business process, therefore it is worthwhile to examine techniques to forecast and avoid upcoming deadline violations.

3.6.3 Predictive Time Management

Predictive time management utilizes the timed graph to make forecasts about the temporal status of currently running processes.

3.6.3.1 Types

- **Remaining Times** These values provide the means for informing customers about the expected remaining process duration and the expected end of the process (e.g. [36]).

- **Upcoming Assignments** E-values enable the time manager to make forecasts about the execution intervals of future activity assignments for involved participants. For instance personal scheduling utilizes this knowledge. It aims at notifying workflow participants about future activity assignments, which increases their possibilities of work-planning in advance, such that they are no longer surprised by new entries in their work-lists (e.g. [37, 75]).

- **Upcoming Deadline Violations** A very important application is the prediction of possible future time violations, e.g. deadline misses (e.g. [34]). The timed graph, enables one to implement a simple but effective escalation warning mechanisms, for example the traffic light model described below.
3.6.3.2 Traffic Light Model

The traffic light model [34] is used to assess the current temporal status of a process. For a simple traffic light two thresholds must be defined (e.g. based on L-values or the available slack time), where the first determines the workflows state change from green to yellow (warn) and the second one determines the state change from yellow to red (alarm). As long as the workflows state is green everything is ok, if the state changes something has to happen.

- In this example the thresholds are based on end-events. The threshold $B_{\text{t_red}} = 6\ p.m.$ indicates a switch from yellow to red and is specified by the calendar mapped LAE-value. The other threshold $B_{\text{t_yel}}$ indicates a switch from green to yellow and lies in the middle between the EPE (3 p.m.) and the LAE-value (6 p.m): $B_{\text{t_yel}} = 4:30\ p.m.$ Assume that $B$ just finished execution at $\text{now} = 5:30\ p.m.$; the temporal status of the process switches to yellow, indicating that a delay during the execution of $C$ may result in a deadline violation. Immediate, proactive evasive actions can be invoked to avoid a deadline violation and change the temporal status back to green again.

The thresholds of the traffic light can be set freely, from risky to conservative adjustments. Of course, it is also possible to employ a more fine grained escalation scheme with more intermediate threshold values. Note that temporal assessment and monitoring of constraint violations must not be constricted to start or end-events. It should rather be performed periodically, in defined intervals. As temporal models provide information for start and end events only, it is necessary to adjust this information according to currently running activity instances, e.g. by subtracting the already elapsed execution time of currently active activities.

3.6.3.3 On Parallel Execution Semantics

The semantics of an and-split demands that each path started by it behaves like an independent sub-process until merged into the main process by an and-join. This implies that the state of execution of such a parallel sub-process is also independent from all its siblings - according to the token-game semantics used in Petri nets several tokens exist, one for each sub-process, and each of them moves independent from the others. Accordingly there are multiple ordering possibilities for the execution of activities in a parallel structures. Figure 3.7 shows a parallel structure. Dependent on the order of activity enactment, a multitude of variants exists, e.g. $A1-B1-A2-C1-B2-C2$, $A1-B1-C1-C2-B2-A2$, etc., which do not even include scenarios with overlapping execution of parallel activities. The more activities and parallel branches exist, the more possible execution scenarios are possible. The above-specified calculation rules for implicit time properties treat each activity that resides on a parallel path as if it were executed in isolation from its parallel siblings, only influenced by precedence constraints. This requires special solutions for time management.
applications, where the point of view is the whole process and not a single activity, as all currently active siblings must be taken into considerations:

- **Process Remaining Duration** The expected process remaining duration is determined by the maximum remaining time of all currently active parallel siblings.

- **Upcoming Deadline Violations** For the traffic-light model it one must take the worst case into consideration, which is determined by the parallel sibling with the minimum L-value.

### 3.6.3.4 Dynamic Adjustments

Scheduling theory distinguishes between static and dynamic approaches [10]. In a static approach a schedule is calculated once at compile time. Whereas a dynamic approach adjusts the schedule periodically – it flexibly responds to changes of the system state, for example caused by a varying work load. For workflow time management the following dynamic techniques can be applied: slack time adjustment and updating the timed graph.

**Slack Time Adjustment** When talking about slack, one has to differentiate between build time and run time. Section 3.5.3.4 already explained how slack can be predicted and distributed at build time. During run time actual activity execution times may vary considerably from the estimated durations. When the actual execution time is less than the estimated duration, slack becomes available. On the other hand, when the execution takes longer than the estimated duration, slack time for future activities will be reduced. Thus predicted slack $v.se$ of a node $v \in \mathcal{V}$ will frequently vary from the actual slack $v.se_{act}$, as $v.se = v.lae - v.epe$ and $v.se_{act} = v.cal.lae - now$.

As this deviation is often caused by varying system load factors the authors of scheduling-based approaches frequently propose to apply *dynamic deadline adjustments*. E.g., [61, 85] adjust the static predicted L-values, which are used in the traffic light model, by taking the current work load into consideration. The strategy *proportional load* distributes available slack in proportion to current work load on agents that are involved in the execution of remaining activities (successors). Average and current system load is compared by means of average and current queue lengths. Another strategy, called *proportional escalation*, [85] favors activities and processes with high escalation costs, by assigning them
a bigger proportion of the available slack time. For further details on run time-oriented slack distribution strategies involving load factors and escalation costs please refer to the above-mentioned publications and the survey in Appendix A.

Updating the Timed Graph A run-time problem of timed graphs is that the origin of E-values is always the start of the process. During process execution decision will be made and actual activity durations will deviate from expected ones. This influences the start times and end times of upcoming activities enormously. The further the execution progresses, the more imprecise the pre-calculated E-values will be. A fire in the darkness is a good metaphor for this problem. The nearer an object is, the clearer it can be spotted, in contrast to distant objects which are diffuse. A person walking away from the light gets into darker areas and is more likely too stumble. To get a better view at the current position another lantern must be lit. Mapping this metaphor to the timed graph means calculating an updated version, starting with the current activity. For further details on and algorithms for the adjustment of a timed graph please refer to Section 5.5.

3.6.4 Proactive time management

Proactive time management builds on the prediction component. The idea is to invoke evasive actions for late process instances in order avoid possible future deadline violations.

3.6.4.1 Types

In [88, 34, 35, 85, 110] the following proactive escalation alternatives, which mainly aim at speeding up the rest of the process, are discussed:

- **Skip optional tasks** During the process specification some tasks may be marked as optional, which means that the process can be finished without executing them; e.g. review activities. If the process is late these optional activities are automatically skipped.

- **Alternative path selection** Different alternatives for one specific part of a process are defined. If the process is late, alternatives with a shorter execution duration will be automatically selected.

- **Early Escalation** The idea is to select certain processes, which will most likely violate their deadlines, for early termination. This saves valuable resources for other process instances and speeds them up.

- **Prioritization** The priorities of late process instances are increased. Tasks belonging to these process instances are privileged in terms of resource allocation. This lets them overtake other process instances and speeds up execution, as the queuing times of their, now privileged, work items are reduced.
- **Parallelizing** Sequential activities, without data flow dependencies between each other, are executed in parallel, which speeds up the process.

- **Task pre-dispatching** Very similar to parallelizing, but restricted to the preparation-phase of future tasks. Some tasks need a preparation phase before they are ready for execution. E.g., it is possible to change the driller head of a CNC-machine, even if the metal plate to be drilled didn't arrive yet.

- **Resource redeployment** The elimination of resource bottlenecks in late process instances will speed them up. This can take many forms, like: adding more resources (e.g. increase manpower), increasing the capacity of an existing resource (e.g. overtime), extending the scope of a role (e.g. temporary advancement of participants), or changing the allocation of tasks (load balancing).

- **Batching** Grouping similar tasks to a batch assigned to a single resource. This decreases the preparation time and eliminates the time that participants usually need for a context switch.

- **Splitting** Sometimes it takes too long to finalize a big batch, therefore it can be split up in several work packages which are parallelized.

- **Escalation sub-process** A sub-process is spawned that performs pre-defined escalation actions, e.g. start compensating actions when a deadline violation already occurred. A proactive sub process could for instance notify stake-holders and renegotiate a new deadline.

All these techniques comprise the potential to speed up processes, but beware: due to unintended feedback effects other process instances may be slowed down, resulting in even more deadline violations than before. The behavior of complex workflow systems with distributed resources and many process instances executed in parallel is hard to foretell and relies on many input parameters. Results achieved with one technique may be perfect for one specific configuration and sub-optimal for another – it is indispensable to conduct experiments on real or simulated systems before approving a certain technique.

In the following two types, *early escalation* and *prioritization*, are discussed in greater detail. They have been applied in diverse simulation scenarios which we conducted to compare and evaluate proactive time management techniques (see Chapter 7).

### 3.6.4.2 Early Escalation

*Early escalation* is a proactive strategy, that decides whether or not to force escalation of late process instances immediately. The objective is to save valuable resources for other processes instances in order to decrease the overall number of late instances as well as costs. The traffic light model is used to detect late process instances which are candidates...
for early termination. The authors of [83, 85] propose to take additional criteria into consideration: it is for example not advisable to cancel instances too early as they might catch up, or to cancel processes where the escalation costs are higher than completion costs. Their algorithm examines a defined number of successor activities and calculates an escalation ratio for each of them. If this ratio, determined from different parameters, like load-dependent predicted completion time and escalation cost, is greater than a value based on the adjusted activity deadlines, then early escalation will be enacted. The ratios incorporate the idea that the higher the cost of an early escalation the more confident one should be that a deadline violation will actually occur.

3.6.4.3 Process Prioritization

Another simple, yet effective, proactive technique is process prioritization. Many workflow systems offer, similar to operating systems, the means to give each process a priority. In terms of resource allocation this priority also decides whether a process instance will be favored or penalized when multiple tasks compete for resources.

Optimization Objectives The main objective of process prioritization is to optimize a certain performance criterion of the system. Scheduling-based workflow literature distinguishes between the following performance measures, e.g. in [85, 120, 4]:

- **Average Turnaround Time** The turnaround time depicts the execution duration of a process instance. As it may vary considerably between different process instances usually the average turnaround time is used as measure.

- **Throughput** This measure depicts the amount of work performed over a period of time, for instance the number of process instances finished per day.

- **Percentage of Late Jobs** Relates the number of late jobs, denoting process instances which violated their deadlines, with timely ones. Optimization of this percentage is reasonable if deadline violations are associated with penalty payments.

- **Mean Tardiness Percentage** Tardiness designates the amount of lateness of one process instance. The tardiness percentage of a late instance relates its actual duration to its allowed duration. The mean tardiness percentage considers all late process instances. Optimization of this percentage is reasonable if the amount for penalty payments rises with increasing tardiness.

Which measure to optimize depends on the objectives one aims to achieve. If a deadline violation always results in the payment of a constant penalty, decreasing the number of late process instances will be more important. If the amount of lateness is of interest then the tardiness will be the measure of choice.
Prioritization Strategies  Giving a process instance a higher priority results in faster (selection and) assignment of adhering work items to participants. This speeds up the execution of privileged instances, but slows down the other instances. For workflow systems and workflow simulation scenarios the following scheduling-based prioritization strategies, also called dispatching rules, have been described [120, 4]:

- **Shortest/Longest Work Time (SWT/LWT)** These strategies prefer the assignment of work items with the shortest/longest expected process duration. SWT is used to optimize throughput, as it tends to favor simpler cases that can be completed quickly, whereas LWT is used to optimize turnaround.

- **Shortest/Longest Remaining Time (SRT/LRT)** They utilize the remaining time and favor work items of process instances which are expected to finish earlier/later than others. LRT can be applied to minimize the mean tardiness, whereas SRT similar to SWT can be used to optimize the throughput.

- **Earliest Deadline First (EDF)** This strategy is also called Earliest Due Date (EDD). It utilizes the overall process deadline such that process instances with earlier deadlines will be favored. It is used to minimize the number of late jobs.

- **First Come First Serve (FCFS)** This strategy is also called First-In First-Out (FIFO). It is easy to implement, does not consider any priorities, and pursues no specific optimization objectives. It is also used to simulate the behavior of queue-oriented machines like printers or the nonselective behavior of human participants.

- **Service In Random Order (SIR0)** This strategy is actually not used in real systems. It is frequently applied in simulation scenarios to mimic the arbitrary selection behavior of human participants.

Workload  The workload of a system is dependent on the arrival rate of process instances (inter-arrival frequency) and the departure rate [90, 4]. Time management prioritization strategies aim at tuning already balanced systems which are not too 'overloaded' or 'underloaded'. In an overloaded system the arrival rate of work items (in the worklists of participants) exceeds the departure rate – this is often caused by bottlenecks, e.g. a central activity can only be processed by one specific participant. In an underloaded system worklists are empty most of the time, therefore items are executed immediately on arrival. Accordingly [90] showed that deadline-oriented selection policies tend to have only minor advantages over conventional strategies in underloaded systems and that they are suboptimal for overloaded systems, as they always try to save already late instances, which results in even more deadline violations and increased tardiness. Therefore time management strategies should be used to optimize the above mentioned measures in already 'balanced' systems.
Reward Policies  In workflow systems tasks can be automatically assigned to participants or agents according to a priority. This is not the case in systems with pull-oriented worklist handlers, where one participant, out of a group of potential performers, selects the work item she wants to execute next [99]. Although work lists can be sorted according to a given priority it is not guaranteed that participants will stick to this order – additional means of motivation must be found. [120] proposes to use reward policies. Participants are rewarded due to an agreed upon performance measure. For instance, the reward is coupled to the number of deadline violations, or to the the number of process instances finished per time unit (throughput), very similar to a piecework system in manufacturing processes.
Chapter 4

A Probabilistic Model for Acyclic Processes

Until now only unconditional structures have been discussed, but when modelling real business processes the usage of conditional structures, like or-splits, is indispensable. These structures cause uncertainties – e.g. uncertain process duration as it depends on the selected path – which cannot be considered in a deterministic model like the timed graph. Thus a probabilistic model must be introduced to capture and represent temporal uncertainty. This chapter describes a probabilistic timed graph, which contains branching probabilities for conditional structures and which represents time properties in the form of time histograms. For the time being this chapter concentrates on non-blocked, acyclic graphs that contain sequential, parallel, and conditional structures.

4.1 Introduction

Uncertainties, with an adversarial influence on the forecast accuracy, are caused by the unpredictable branching behavior of conditional structures and the often unforeseeable execution duration of a task at run time. Existing literature on workflow time management proposed several techniques to cope with that uncertainty, but none of them comes without major drawbacks.

4.1.1 Problem Statement

4.1.1.1 Conditional Structures

A decision made at or-splits often depends on data and conditions that cannot be evaluated in advance, e.g. a participant has to enter some data, which is later used to evaluate a branching condition, or a user has to choose one out of several successor tasks. This complicates the build time calculation of valid execution intervals enormously, as it is for instance not possible to unambiguously determine the start time of activities after a conditional structure. Consider the following example: during the forward calculation of the COND-process, presented in Figure 4.1, a problem arises at the or-join. In contrast to an and-join, which waits until all predecessors are finished, the or-join will be started as
soon as one of its predecessors is finished. Whether this predecessor will be B1, B2 or B3 depends on a decision made during run time. Therefore $OJ$ is not ambiguously determinable at build time – it could be 7, 4 or 10 – and even during run time this value will not be known time until a choice is made. In analogy, a problem arises also for the backward calculation of L-values, but now at the or-split. In the example the path via B3 would even result in a deadline violation. Analogously, it is not possible to unambiguously determine the rest times of activities, as well as the overall process duration.

![Figure 4.1: A Conditional Control Flow Structure](image)

4.1.1.2 Activity Durations

Apart from that, the run-time duration of an activity will most certainly differ from its expected duration – the reasons are diverse.

- **Dynamic assignment of agents**: many workflow systems feature the possibility to dynamically assign agents to certain tasks, often in a role-based fashion. The execution duration will most likely differ from participant to participant, as for instance novices will need more time to finish tasks.

- **Task selection**: in pull-oriented systems participants are allowed to select tasks out of a list of task, which are usually assigned to a participant with a certain role. The queuing time of a task – which is the time it resides in the worklist – may depend on personal preferences of this participant. If the participant has several roles she will probably execute similar tasks in a batched fashion, in order to reduce context-switching delays.

- **Task prioritization**: tasks from process instances with low priorities will remain in worklists longer than those from processes with high priorities. Usually it is possible to dynamically adjust priorities during process execution.

- **Case-dependent duration**: the duration of a task may vary considerably depending on the specific case, e.g. the time spent to handle a claim depends on the importance of the customer.

- **System work load**: high work load produces more work items in the worklists of participants, therefore waiting or queuing times will be longer.
4.1 Introduction

- **Work load of participants:** workflow participants will often have duties and responsibilities which do not stem from workflow task assignments. Depending on the load produced by these 'outside-tasks', sometimes more and sometimes less time will be available to execute workflow tasks.

Therefore a scalar average value for the expected activity duration falls short, as it is to imprecise, especially for prediction purposes. Consider the following (rather extreme) example: an activity takes either 2 or 1000 minutes, each with a probability of 50%. The arithmetic mean of the duration is therefore \((2 \times 0.5 + 1000 \times 0.5) = 501\). If, what we expect, future instances of this activity behave like their past executions, then the average duration will always deviate 499 minutes from the actual value.

4.1.2 Existing Methods of Resolution

In time management literature several techniques have been proposed which aim at dealing with temporal uncertainty.

4.1.2.1 Conditional Structures

- **Treat like Parallel Structures** In many time management approaches it is common practice to consider only the worst case, which is the path with the longest duration. According to this view conditional structures are treated like parallel structures (e.g. [122, 6]). This is quite suitable for constraint satisfaction techniques, as they only have to check if time constraints are satisfiable at all. It is also applicable for the calculation of the maximum process duration, which is determined by the longest path (e.g. [98]). And especially in time critical (hard real-time) systems it makes sense to calculate only maximum E-values and minimum L-values – for instance in a nuclear power plant the knowledge about critical values is of utter importance [108].

- **Splitting Instance Types** Especially in techniques originating from soft real-time scheduling it is common practice to deal with every possible process instance in isolation. An instance type consists only of sequential and parallel structures, hence precedence and timing of activities is deterministic. During build time the timed graph must therefore be calculated for each identified instance type (e.g. [6, 72]). This technique is especially suitable for constraint satisfaction at build time. Nevertheless, some approaches also apply it during run time to enable predictive features. To achieve this the system has to switch between instance types as soon as a decision has been made.

- **Deterministic Timing – Average Values** A very simple, though questionable, approach is to calculate average time values, based on the expected branching behavior of or-splits. For instance [120] introduces branching probabilities, weights time val-
ues at or-splits according to these branching probabilities, and show how to calculate the expected process (remaining) duration.

- **Intervals** The most common technique is the usage of intervals, as for instance in [34, 76]. They basically determine \([\text{min}, \text{max}]\)-intervals for all implicit time constraints. Consider the following example, based on the COND-process: 

\[
\text{Start.} = \text{Start.} = [0, 0], \text{A.} = \text{O.} = \text{B1.} = \text{B2.} = [3, 3], \text{B1.} = [7, 7], \text{B2.} = [4, 4], \text{B3.} = [10, 10].
\]

The EPS-interval of the or-join is defined by the minimum and the maximum EPE-interval bounds of its predecessors: 

\[
\text{O1.} = [4, 10], \text{O1.} = [4, 10], \text{C.} = [4, 10], \text{C.} = \text{End.} = \text{End.} = [9, 15].
\]

Within these intervals one may reason about different temporal properties of the process, such as, activity C will presumably be started between 4 and 10, or, the process will presumably last between 9 and 15 time units. The backward calculation of LAE and LAS-intervals works analogously.

4.1.2.2 Activity Durations

Accordingly, to capture duration uncertainties, some approaches introduce \([\text{min}, \text{max}]\)-intervals for activity durations (e.g., [78, 20]). This kind of representation is also very frequently used in temporal constraint networks, which view a duration as a limited duration constraint (e.g., [50, 6]). Duration intervals are easily integrated in a model that supports an interval-representation for implicit time constraints. Additionally continuous distribution functions – for instance normal or beta distributions – are used to describe durations (e.g., [88, 18]). However these approaches do not actually integrate the distribution in the calculation of implicit time properties; they apply it to generate a deterministic (average) expected duration for further usage.

4.1.3 Some Problems Remain

4.1.3.1 Conditional Structures

Although the above-mentioned techniques and representations may be suited for specific time management objectives, they all fall short when it comes to predictive features like forecasting upcoming activities or assessing if the process instance is late, especially for processes that contain conditional structures. For each approach several shortcomings can be identified.

- **Treat like Parallel Structures** This technique is inhibited by two severe shortcomings. First, as they do not distinguish between conditional and parallel execution, they have to take every possible activity assignment into consideration, which may result in over-constrained schedules which results in false forecasts about resource-utilization as in e.g., in [21]. Second, considering only worst cases, as the path with
the longest duration determines implicit time constraints, will often be too strict for realistic forecasts.

- **Splitting Instance Types** To treat every possible process instance in isolation is insufficient, because the existence of multiple conditional structures in a process model quickly results in a vast number of instance types – [88] even speaks of *complexity explosion*, as the number of possible alternatives grows exponentially with the number of or-splits. Additionally the workflow system has to switch between instance types as soon as a decision has been made. This reduces predictive capabilities tremendously as it is basically not possible to look past the next decision point. Some authors propose to simply stop the forward examination at or-splits (e.g. [61]) and to recalculate implicit time properties after each decision (e.g. [12]), which is rather unsatisfying. Even for constraint-satisfaction-techniques splitting of instance types does not come without problems, as according to [29], a set of time constraints may be unsolvable if there is no process instance that does not violate at least one time constraint, even when every single constraint is solvable in isolation

- **Deterministic Timing – Average Values** Techniques which utilize the calculation of average values mainly aim at predicting process durations and activity rest times, based on the average duration of conditional blocks. This does not allow the prediction of upcoming activities which are nested inside a conditional structure. Thus some authors propose guessing or simulate the route a job will follow [4, 59], according to given stochastic information, which naturally generates problems if the guess is wrong.

- **Intervals** Although this technique is superior to the other ones, it still is inhibited by some drawbacks. In approaches with an interval-representation implicit time constraints are solely determined by the paths with the longest and the shortest duration respectively. If these two cases are rarely executed, the valid execution interval of activities will be unnecessarily constrained, which may result in suboptimal forecasts and schedules. Additionally, with every conditional structure the minimum and maximum of the valid execution interval will further drift apart, temporal information will get fuzzier with every or-split.

### 4.1.3.2 Activity Durations

Scalar average values, intervals, as well as continuous distribution functions fall short when it comes to representing the duration of an activity; especially if this duration could be represented as a scattered distribution function with multiple peaks – which in particular will be the case for complex activities that hide whole processes containing conditional structures.
To eliminate these drawbacks this chapter introduces a probabilistic model, based on empirical or estimated information about the expected branching behavior of a process, where activity durations as well as implicit time constraints are represented by means of arbitrary discrete distributions, called time histograms, which aim at improving the forecasting capabilities of the model.

### 4.2 Related Work

Table 4.1 provides an overview of existing time management approaches, that were already introduced in Section 3.2. Note that Jasper/Zukunft and Bussler are not considered, as the corresponding publications do not provide enough details for a comparison. The table compares structural support and representation of temporal information based on the following criterions:

- **Blocking**: in this chapter only acyclic structures are considered, therefore only the conformance classes full-blocked and non-blocked are distinguished. Full-blocked models allow only for proper nesting of according split and join-elements, whereas non-blocked structures allow arbitrary combinations as long as the process is still sound.

- **Structures**: supported control flow structures are denoted by +, unsupported structures by -, and partial support by -/+ . Partial support means that the corresponding

<table>
<thead>
<tr>
<th>Authors</th>
<th>Blocking</th>
<th>Structures</th>
<th>Temporal Information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>acyclic</td>
<td>Seq</td>
<td>Par</td>
</tr>
<tr>
<td>1 Kao, Garcia</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>2 Jasper, Zukunft</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 Haimowitz, et al.</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>4 Panagos, Rabinovich</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>5 Eder, et al.</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>6 Eder, Panagos, et al.</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>7 Olivera, Marjanovic</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>8 Bussler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 Zhao, Stohr</td>
<td>full-blocked</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>10 Kafeza, Karlapalem</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>11 Zhuge, et al.</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>12 Bettini, et al.</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>13 Combi, Pozzi</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>14 Son, Kim, et al.</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>15 v.d.Aalst, et al.</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>16 Li, Fan</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>17 Baggio, et al.</td>
<td>full-blocked</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>18 Lu, Sadiq et al.</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

**Table 4.1: Comparison of Structural Support**
4.2 Related Work

approach applies a suboptimal technique to resolving temporal uncertainties for con-
ditional structures, namely: treat like parallel, split instances types, or calculation of
average values.

- **Explicit temporal information**: these columns contain information about explicit tem-
  poral information. Dur denotes the representation of activity durations, which can be
deterministic/average (det), interval-based (int), or probabilistic (prob). The column
Const differentiates between models that allow a single process deadline (dead) or
advanced time constraints (adv).

- **Implicit temporal information**: these columns list implicit time properties and con-
  straints, that are supported by the corresponding approach, along with their type
  of representation (det, int, prob). Start denotes earliest possible start times or ready
times (in scheduling approaches), End denotes latest allowed end times or due dates
  (in scheduling approaches), and Rest denotes remaining times.

As on can see time management approaches which utilize probabilistic information are
rare. As already mentioned continuous distribution functions have already been applied to
capture the duration of an activity. Eder et al. [88] proposed the use of a beta distribution.
In Baggio et al. [4] durations are taken from a uniform distribution over a given interval.
Son and Kim et al. [18, 98] describe each activity as an independent queuing system,
where service request arrivals form a Poisson process and service times are described by
means of an exponential distribution. However all approaches have one thing in common,
they use distributions solely for determining the expected (average) duration of activities
which is then used for the calculation of further non-probabilistic time properties.

Zhao and Stohr [120] were among the first to integrate empirical knowledge about the
branching behavior of the process. They introduced branching probabilities to describe
the likelihood of selecting one specific branch after an or-split. This was no revolutionary
new concept as, for instance in the areas of business process simulation or re-engineering,
probability distributions that describe the expected branching behavior were already a
well-established concept [69]. Nevertheless, it was new for time-aware workflow applica-
tions, but they still rely on an average-value representation for time properties. The same
has been proposed by Baggio et al. [4] – they use branching probabilities to ‘guess’ the
route a job will follow. Based on these guesses, the problem can be reduced to a deter-
ministic problem – a schedule for process instance which does not contain conditional
structures. Their approach has the same shortcomings as every technique relying on the
examination of one specific instance type.

Van der Aalst et al. [111] introduce discrete time stochastic Petri nets which can be applied
to workflows. The service time of a transition is represented as arbitrary discrete random
variable, called service density – as a matter of fact this representation is similar to the
duration histogram used in this thesis. The branching behavior of conditional blocks is
described by a Bernoulli-distributed random variable. The authors also state that it is possible to compute an arrival function for each single place in the net, similar to the E-histograms described in this thesis. However, this approach aims at build time analysis of process performance measures. It does not consider explicit deadlines, therefore it is not suited to determining implicit time constraints like the latest allowed end time, as well as other backward-calculated time properties, like remaining times.

Parts of the results, which are presented in this chapter, have already been published by Eder, Pichler, and various co-authors. In [36] we proposed the usage of duration histograms and branching probabilities to calculate probabilistic remaining times. [37] shows how to calculate a probabilistic timed graph, containing E and L-histograms, which can be utilized to generate personal schedules. In [8] and [9] we extended the probabilistic model with fixed-date constraints and described how to predictively schedule processes in order to avoid unnecessary delays. All techniques are based on a full-blocked workflow model, that allows the specification of sequential, conditional and parallel structures.

For further discussions on the listed publications please refer also to the survey in Appendix A.2.

4.3 A Probabilistic View on Processes

4.3.1 Branching Probabilities

The main idea of the probabilistic approach is to improve the predictive capabilities of the workflow systems, based on stochastic information about the above mentioned uncertainties. This stochastic information may come from expert or empirical knowledge. The basic concepts are best explained by means of an example. Figure 4.2 shows a simple workflow, that consists of two conditional branches after E and L. At run time only one out of 8 possible instance types will be executed. The duration of instance types is calculated by summing up the durations of all participating activities. Assume that node durations, for explanatory reasons still represented as scalar values, and branching probabilities for or-splits stem from empirical knowledge. Furthermore assume that branching decisions are
independent from each other – thus it is possible to calculate execution probabilities for each instance type by simply multiplying the branching probabilities of or-splits on the according path. This is, e.g., \(0.9 \times 0.48 = 0.432\) for the path from \(A\) via \(B\) and \(E\) to \(I\). The left hand side of Figure 4.3 lists all possible instance types (control nodes omitted), along with their probabilities and their summed up durations.

<table>
<thead>
<tr>
<th>Instance Types</th>
<th>Probability of discrete random variable (d)</th>
<th>Cumulated probability of discrete random variable (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACDEI</td>
<td>0.048 15</td>
<td>0.048 15</td>
</tr>
<tr>
<td>ACDEHI</td>
<td>0.002 19</td>
<td>0.048 15</td>
</tr>
<tr>
<td>ABDDEI</td>
<td>0.432 22</td>
<td>0.002 19</td>
</tr>
<tr>
<td>ABDEHI</td>
<td>0.018 26</td>
<td>0.432 22</td>
</tr>
<tr>
<td>ACDIF</td>
<td>0.048 29</td>
<td>0.018 26</td>
</tr>
<tr>
<td>ACDFI</td>
<td>0.002 29</td>
<td>0.050 29</td>
</tr>
<tr>
<td>ACDGI</td>
<td>0.432 36</td>
<td>0.450 36</td>
</tr>
<tr>
<td>ABDGI</td>
<td>0.018 36</td>
<td>1.000 36</td>
</tr>
</tbody>
</table>

Figure 4.3: Instance Types and Duration Distributions

4.3.2 Time Histograms

4.3.2.1 Regular Time Histograms

Based on the information provided above a discrete probability a distribution function can be applied that describes the duration of the whole workflow: \(P_D(d_i) = P(D = d_i)\), which specifies the probability that the discrete random variable \(D\) equals some value \(d_i \in \mathbb{N}\). The sum of all probabilities must be 1: \(\sum_{i=1}^{N} P_D(d_i) = 1\). The table in the middle of Figure 4.3 shows the correlation between instance types and the values of the distribution. Note that the execution probabilities of instance types with equal durations must be merged. This also indicates that only different time values are of interest, not every possible instance type. This eliminates the necessity of storing information about different instance types. Figure 4.4 illustrates why the name *time histogram* has been chosen. The distribution can be charted as a histogram where every tuple describes one bar.

4.3.2.2 Cumulated Time Histograms

One of the main purposes of time management is to express, process and interpret queries on different time properties. Some examples are a) the expected duration of a workflow with a given probability, b) the probability that an activity will start at a given point in time, or c) the remaining duration from the current activity to the end of the workflow with a given probability. To express these queries it is necessary to cumulate the values. The cumulated distribution function, also called probability mass function, \(F_D(d_i) = F(D \leq d_i) = \sum_{j=1}^{i} P_D(d_j)\) is used to indicate the probability that the duration is less or equal \(d_i\). The right hand table in Fig. 4.3 shows such a cumulated histogram which is interpreted
as follows: There is a 4.8% chance that the workflow will not last longer than 24, and a 48.2% (=0.048 + 0.432) chance that the workflow will not last longer than 27, and so on. Finally we conclude that there is a 100% chance that the workflow will not last longer than 41. Based on this function it is therefore possible to determine the expected duration for a given minimal probability, as well, as the expected probability for a given duration.

4.3.2.3 Types of Time Histograms

A time histogram describing a duration is also called duration histogram, or D-histogram [36]. As the name implies, it describes the duration of an elementary activity, a complex activity, or a whole process. It can be extracted from workflow logs by examining executions of past instances. If, due to the lack of data, an empirical method is not applicable, expert estimations can be used. For complex activities and processes it is also possible to determine them based on the structure, elementary activity durations, and branching probabilities. Furthermore E and L-values of nodes have, similar to the process duration, multiple possible values with different probabilities. Consider the following examples:

- Because of the conditional structure preceding activity D two routes with different durations must be considered, thus the earliest possible start time is no longer unambiguously determinable. According to the branching probabilities, D starts at 14 with a probability of 0.9 and at 7 with a probability of 0.1.

- Based on an assumed process deadline of $\delta = 40$ and considering possible routes from the end of the workflow, activity D has four latest allowed end times: 36 with 0.48, 22 with 0.48, 22 with 0.02 and 32 with 0.02. Applying the above explained
aggregation, the two value-pairs holding an equal time value of 22 can be aggregated, yielding 22 with a probability of 0.5.

Thus, implicit time constraints are representable as histograms. In analogy to D-histograms they are called E-histograms and L-histograms. The same applies for remaining and slack times which are represented by R and S-histograms. Note that in contrast to other types, cumulated L-histograms are calculated and interpreted differently. For this and further aspects of histogram cumulation and interpretation please refer to Section 5.1. Another extremely important feature is called histogram compression. It is used to avoid complexity explosion, by reducing the size of histograms (see Section 5.1.4).

4.3.3 Basic Operations on Time Histograms

Definition 9 (Time histogram): A time histogram $H$ is a set of tuples $(p, t)$, where $0 < p \in \mathbb{R}$ is the probability and $t \in \mathbb{N}$ is the according time value.

Definition 10 (Cardinality): The cardinality $|H|$ of a time histogram $H$ is determined by the number of tuples in $H$.

Definition 11 (Histogram property: aggregated): A time histogram $H$ is aggregated if there are no two elements $(p_i, t_i), (p_j, t_j) \in H$, where $i \neq j$ and $t_i = t_j$.

Definition 12 (Histogram property: valid): A time histogram $H$ is valid if $\sum_{i=1}^{|H|} p_i = 1$ for all $(p_i, t_i) \in H$.

To adhere to the definition of a discrete probability function a time histogram must be aggregated, thus it must hold only unique time values, and it must be valid, thus the probabilities must sum up to one. The duration of the whole process in figure 4.4 is defined by a valid and aggregated time histogram $H = \{(0.048,15), (0.002,19), (0.432,22), (0.018,26), (0.050,29), (0.450,36)\}$ with a cardinality of $|H| = 6$.

Definition 13 (Weight): A time histogram $H_1$ is weighted by a probability $w$ resulting in $H_2$ as follows: $H_2 = H_1 \ast w = \{(p \ast w, t) \mid (p, t) \in H_1\}$.

Definition 14 (Addition of a scalar): The scalar $k$ is added to $H_1$ resulting in $H_2$ as follows: $H_2 = H_1 + k = \{(p, t + k) \mid (p, t) \in H_1\}$.

Definition 15 (Subtraction of a scalar): The scalar $k$ is subtracted from $H_1$ resulting in $H_2$ as follows: $H_2 = H_1 - k = \{(p, t - k) \mid (p, t) \in H_1\}$.

Definition 16 (Negation): $H^-$ denotes a negated time histogram $H$ such that $H^- = \{(p, -t) \mid (p, t) \in H\}$.

As time histograms are based on regular sets, the set operations ($\cap, \cup, \setminus$) can also be applied. For a better understanding of the presented basic operations consider the following examples:
1. A histogram: $H_1 = \{(0.2,12), (0.3,12), (0.5,13)\}$

2. Another histogram: $H_2 = \{(0.5,12), (0.5,13)\}$

3. Weight: $H_3 = H_2 \times 0.5 = \{(0.25,12), (0.23,13)\}$

4. Add scalar: $H_4 = H_2 + 5 = \{(0.5,17), (0.5,18)\}$

5. Subtract scalar: $H_5 = H_4 - 5 = \{(0.5,12), (0.5,13)\}$

6. Negation: $H_5^- = \{(0.5,-12), (0.5,-13)\}$

7. Yet another histogram: $H_6 = \{(0.5,12), (0.25,20), (0.25,22)\}$

8. Set union: $H_7 = H_1 \cup H_6 = \{(0.5,12), (0.5,13), (0.25,20), (0.25,22)\}$

9. Set intersection: $H_8 = H_1 \cap H_6 = \{(0.5,12)\}$

10. Set minus: $H_8 = H_1 \setminus H_6 = \{(0.5,13)\}$

Given a valid time histogram, the operators weight, set union, set intersection and set minus may yield a non-valid time histogram as result (see examples 3, 8, 9 and 10). In contrast to this, the operations add scalar, subtract scalar and negate never change any histogram properties (see examples 4, 5 and 6).

4.3.4 Defending the Model

The above-specified model assumes dependency of events and uses discrete distributions to represent temporal properties. Naturally the question arises why this model has been chosen?

**Independence of Events** All considerations presented above are made under the assumption of statistical independence of events. This means that two (successive) events are independent from each other – for instance two consecutive decisions on a path with or-splits. Now one could state that these events are not always independent from each other, that, in contrary, quite the opposite is true. It is not hard to come up with examples, as certain decisions made by users often influence subsequent run-time evaluated automatic decisions. A similar dependency is based on data, as for example data entered or altered in prior tasks may influence the decision at an or split. Yet another form of dependency might be temporal. Consider the following sequential process: the first editor is responsible for shortening a new novel. She does this very thoroughly and shortens it down to a small fraction of its original length. This took her very long. Then the second one gets the novel and reviews it rather fast, as it is already shortened. The amount of time spent on the predecessor directly influences the duration of the successor.
However, even if it would seem reasonable to use conditional probabilities at first glance, there are several good reasons not to do this. First, it is hard to extract this information from the workflow log, as it will often not be decidable if these are real dependencies or mere coincidences, which leaves this rather complex task to the workflow designer. Second, the usage of conditional probabilities implies that we would have to store very huge complex structures, which contain knowledge about different instance types, or at least prior decisions made at or-splits. This further deprives us of the possibility to aggregate values, as well as cumulating, compressing, comparing or applying simple algebraic operations on them. Additionally, it would not be possible to query these structures in a simple way, as the interpretation of results would be rather complicated; e.g. "activity X will start before 10, with a probability of 90%, but only under the condition that ...". To make a long story short: the advantage gained, which is getting more accurate statements, does not justify the disadvantages introduced by such a model.

**Discrete Distributions** As already mentioned above, there are several ways to store durations and branching behavior by means of continuous, for instance beta or exponential, distributions. They can be used to describe such facts in a very compact way, just by specifying few basic properties. Nevertheless they have one major drawback: they are ill-suited to representing arbitrary distributions – for instance scattered distributions with multiple peaks which will in particular be needed for the representation of durations and other time properties stemming from processes which contain conditional structures. This justification can also be found in [111], which applies discrete time domains for the calculation of durations of blocked structures in a given process. They discuss the usage of other related techniques, for instance Markov-chains which are restricted to certain continuous distribution types which defy exact analysis. They also state that the use of discrete distributions is not a real limitation, since a fine-grained discrete time domain can be used to approximate a continuous time domain. On the other hand they mention that the algorithms are potentially inefficient as distributions over the discrete time domain are represented explicitly, hence tend to get very large. To counter that we introduced histogram compression techniques, as described in a Section 5.1.4.

4.4 Probabilistic Workflow Graphs

4.4.1 Probabilistic Extended Workflow Graph

The probabilistic extended workflow graph (PE-Graph) augments the basic workflow graph, as presented in Section 2.8, with explicit probabilistic time properties and probabilistic information about the branching behavior. As in the basic extended graph, this information may come from expert estimations or stem from empirical knowledge extracted from the workflow history.
Definition 17 (Probabilistic Extended Workflow Graph): A probabilistic extended workflow graph \( G_{PE} = (V, E, \delta) \) consists of a set of probabilistic extended nodes \( V \) (vertices), a set of (directed) edges \( E \), and a process deadline \( \delta \).

Definition 18 (Probabilistic Extended Node): A node \( v.t \in V \) in a probabilistic extended workflow graph represents an activity or control node, where \( v.t \in T \) depicts its type, \( v.D \) is its duration histogram and \( v.B \) describes its branching behavior.

The D-histogram contains probabilistic information about the expected duration of a node – as already explained above. The branching behavior contains branching probabilities between one specific node and all its successors.

Definition 19 (Branching Behavior): For a node \( v \in V \) with \( n = |v.Succ| \) successors \( v.b = \{(s_1, p_1), \ldots, (s_n, p_n)\} \) describes the branching behavior of \( v \), where \( 1 \leq p_i \leq n \) is the branching probability \( p_i \) to a successor \( s_i \in n.Succ \). The branching probability \( p_i \) is the (expected) likelihood for a node \( s_i \) to be executed immediately after its structural predecessor \( v \).

To guarantee sound temporal models the branching behavior must adhere to some basic conditions which are dependent on the node type \( v.t \) of a node \( v \):

- if \( v.t = \text{or-split} \) then \( \sum_{i=1}^{n} p_i = 1, \forall (s_i, p_i) \in v.b \)
- else if \( v.t = \text{and-split} \) then \( p_i = 1, \forall (s_i, p_i) \in v.b \)
- else \( p = 1, |v.B| = 1 \)

According to these conditions only the branching behavior of or-splits holds probabilities \( p < 1.0 \), e.g., for OS2 in Figure 4.2: \( OS2.b = \{(E,0.48), (F,0.48), (G,0.02), (H,0.02)\} \).

4.4.2 Execution Probabilities

Definition 20 (Execution Probability): The execution probability \( v.x \) of a node \( v \in V \) represents the likelihood for its execution.

According to the soundness-property the execution probability of the start and the end-node must be 100%. Consider activity \( C \) in figure 4.5: the execution probability of \( A \) is 20%, because only 20% of all instances will execute the route via \( A \), and the execution probability of \( OJ1 \) is 100% as every possible execution will pass this node. These examples are very intuitive because the process is block-structured, but when it comes to valid non-blocked processes, like the one in figure 4.6, the determination of execution probabilities ceases to be that straightforward. Algorithm 4.1 shows how to calculate the execution probability \( v.x \) for each node \( v \in V \), by applying node-type-specific operations in a forward topological order. The algorithm in a nutshell: the execution probability of an or-split successors is

---

\(^1\)Already defined in Section 2.8.
4.4 Probabilistic Workflow Graphs

![Figure 4.5: Execution Probabilities](image)

![Figure 4.6: Execution Probabilities in a Non-blocked Structure](image)

Calculated by weighting the execution probability of the or-split with the respective branching probabilities. All other nodes receive the execution probability of their predecessors(s), where and-joins are a little bit special, as they merge several predecessors (see Figure 4.7 for an example). In this case it is important to notice that every predecessor of an and-join must have the same execution probability. Different execution probabilities indicate that branches to be merged at the and-join may be executed conditionally. This also implies that they adhere to different instance types. This is only possible in unsound structures, as displayed in Figure 4.8. When applying the calculation operations for execution prob-

**Algorithm 4.1 Execution Probability Calculation**

```
Input: Probabilistic extended graph G_{PE} = (V, E)
1: for each v ∈ V, in a forward topological sort order do
2:   if v.t = start then
3:     v.x = 1.0
4:   else if ∃ pred⇒v ∈ E and pred.t = or-split then
5:     v.x = pred.x * p, ∃(v, p) ∈ pred.b
6:   else if v.t = or-join then
7:     v.x = ∑ pred.x, ∀ pred ∈ v.Pred
8:   else if v.t = and-join then
9:     v.x = pred.x, ∀ pred ∈ v.Pred
10:  else
11:     v.x = pred.x, ∃ pred ∈ v.Pred
12: end if
13: end for
```
abilities a problem occurs at the and-join, as nodes with different execution probabilities meet. Additionally $OJ_2$ has an execution probability of 125%, which is simply impossible. Therefore, execution probabilities may also be used as indicator for the soundness of a workflow. But note that the validity of execution probabilities is not a sufficient indicator to check the soundness, as there may exist probabilistically valid structures, which are not sound. In fact it is quite easy to come up with an example, e.g., by changing the branching probabilities of $OS_2$ to 25% and 75%, such that the execution probabilities of $OJ_1$ and $OJ_2$ sum up to 100%.

4.4.3 Forward and Backward Branching Probabilities

Branching probabilities can not be used in a straightforward manner for weighting temporal information for E and L-histograms. E-histograms are temporally influenced by preceding nodes, where the 'degree of temporal influence' depends on node types and branching probabilities. Considering the example in Figure 4.5: the degree of influence of $Start.epe$ on $OS_1.eps$, or for $OS_1.epe$ on $OS_2.eps$, and also for $OS_2.epe$ on $A.eps$ is 100% as there exists exactly one possible route to these nodes. This is not the case at the join
node: OJ2.eps is influenced by A.epe and C.epe, therefore it is a necessity to weight these values accordingly. L-histograms are influenced by successors, therefore weighting must take place at split nodes. Accordingly, during forward and backward calculation different probabilities, called forward and backward branching probabilities, must be used. These probabilities are applied on each edge and they are determined on the basis of execution probabilities and branching behavior of the connected nodes.

4.4.3.1 Forward Branching Probabilities

Definition 21 (Forward Branching Probability): A forward branching probability $p_{\text{pred} \rightarrow \text{succ}}$ where $\text{pred} \Rightarrow \text{succ} \in \mathcal{E}$, specifies the likelihood that, during process execution, when $\text{succ}$ starts, $\text{pred}$ has been executed before it.

Calculation is based on the types and execution probabilities of two connected nodes: the predecessor $\text{pred}$ and its successor $\text{succ}$.

- if $\text{succ}.t = \text{or-join}$ then $p_{\text{pred} \rightarrow \text{succ}} := \frac{\text{pred}.x}{\text{succ}.x}$
- else $p_{\text{pred} \rightarrow \text{succ}} = 1$

Figure 4.9 shows an example that is based on the process displayed in Figure 4.6. The forward branching probabilities are attached to the corresponding edges. $p_{F \rightarrow OJ2} = 0.125$ indicates that in 12.5% of all cases a token will come from activity $F$, therefore the degree of temporal influence of $F$ on the E-values of $OJ2$ is 12.5%. Please note that the displayed probabilities $p_{E \rightarrow OJ1}$ and $p_{C \rightarrow OJ1}$ are rounded.

4.4.3.2 Backward Branching Probabilities

Definition 22 (Backward Branching Probability): A backward branching probability $p_{\text{pred} = \text{succ}}$, where $\text{pred} \Rightarrow \text{succ} \in \mathcal{E}$, specifies the likelihood that, during process execution, $\text{succ}$ will be the successor of $\text{pred}$.

A backward branching probability $p_{\text{pred} = \text{succ}}$ specifies the likelihood, that after finishing the node $\text{pred}$, $\text{succ}$ will be executed directly after it. Calculation is based on the types and execution probabilities of nodes $\text{pred}$ and $\text{succ}$.
if \( \text{pred.t} = \text{or-split} \) then \( \frac{p_{\text{pred} \Rightarrow \text{succ}} = \frac{sucx}{predx}} \)

else \( p_{\text{pred} \Rightarrow \text{succ}} = 1 \)

Figure 4.10 shows an example that is based on the process displayed in Figure 4.6. The backward branching probabilities are attached to the respective edges (displayed with arrows reversed). \( p_{OS1} = B = 0.5 \) indicates, that in 50% of all cases the route via \( B \) will be chosen. Therefore the degree of temporal influence of \( B \) on the L-values of \( OS1 \) is 50%. Note that backward probabilities can also be determined by directly applying the branching probabilities specified in the branching behavior of a node, such that \( p_{\text{pred} \Rightarrow \text{succ}} = p \), where \( (\text{succ}, p) \in \text{pred.b} \).

4.4.4 Probabilistic Timed Workflow Graph

The probabilistic timed workflow graph (PT-Graph) augments the probabilistic extended graph, as presented above, with time properties and constraints. It is used to calculate all implicit time constraints and properties.

**Definition 23 (Probabilistic Timed Workflow Graph):** A probabilistic timed workflow graph \( G_{PT} = (\mathcal{V}, \mathcal{E}, \delta) \) consists of a set of probabilistic timed nodes \( \mathcal{V} \) (vertices), a set of directed edges \( \mathcal{E} \), and a process deadline \( \delta \).

The deadline can be specified as a histogram, consisting of several thresholds with probabilities. In the examples used throughout this chapter, for reasons of simplicity, a simple scalar deadline is used.

**Definition 24 (Probabilistic Timed Node):** A node \( v.t \in \mathcal{V} \) in a probabilistic timed workflow graph represents an activity or control node, where \( v.x \) depicts its execution probability, \( v.t \in \mathcal{T} \) its type, \( v.d \) its D-histogram, \( v.eps \) its EPS-histogram, \( v.epe \) its EPE-histogram time, \( v.las \) its LAS-histogram, and \( v.lae \) its LAE-histogram. The histograms \( v.ss \) and \( v.rs \) denote slack and remaining time at the start of \( v \), and \( v.se \) and \( v.re \) slack and remaining time at the end of \( v \).

Forward and backward branching probabilities, as explained above, are determined based on the structure and the specified branching behavior of nodes.

\(^2\)Already defined in Section 2.8.
4.5 Calculation of the Probabilistic Timed Graph

Based on a PE-graph with D-histograms for the duration of each node, the branching behavior of each node, and an overall deadline, the probabilistic timed graph can be calculated. Similar to the basic algorithm, described in Chapter 3, all implicit time constraints and properties are determined in a forward or backward topological order respectively. The operations will be explained based on a simple example presented in Fig. 4.11. The overall workflow deadline is $\delta = 19$. According to the branching behavior specified for $OS$, the forward and backward branching probabilities are: $p_{C\rightarrow Oj} = p_{OS\rightarrow C} = 0.3$ and $p_{D\rightarrow Oj} = p_{OS\rightarrow D} = 0.7$, and 1.0 for all other edges. The node-durations are specified by the following D-histograms, where the duration of control nodes is assumed to be 0.

$$\begin{align*}
\text{Start.d} &= \text{End.d} = \{(1.0,0)\} \\
\text{OS.d} &= \text{Oj.d} = \text{AS.d} = \text{Aj.d} = \{(1.0,0)\} \\
\text{A.d} &= \{(0.7,3), (0.3,5)\} \\
\text{B.d} &= \{(0.5,3), (0.5,5)\} \\
\text{C.d} &= \{(1.0,2)\} \\
\text{D.d} &= \{(1.0,4)\} \\
\text{E.d} &= \{(0.8,3), (0.2,5)\} \\
\text{F.d} &= \{(1.0,4)\}
\end{align*}$$

4.5.1 Forward Calculation of E-histograms

To determine EPS and EPE-histograms of every node $v$ the forward calculation rules specified in table 4.2 have to be applied to each node in a topological order according to the node type $v.t$, where $\text{pred}$ denotes a predecessor $\text{pred} \in v.\text{Pred}$. For the sake of an improved overview the forward calculation is displayed in Fig. 4.12. The arrows indicate the order of calculation operations, the boxes represent EPS, EPE, LAS, and LAE-histograms, and the
labels of rounded boxes indicate which histogram operation has been applied. As no delay or waiting times between nodes are considered, the EPE-histogram of a node is equal to the EPS-histogram of its predecessor, except for join nodes where multiple edges meet.

Figure 4.12: Forward Calculation of the PT-Graph

4.5.1.1 Adding the Duration and Calculation of Sequences

Initially the EPS-histogram of Start must be set to \(\text{Start}.eps = \{(1.0,0)\}\). The EPE-histogram of each node \(v\) is determined by adding its D-histogram (duration) to its EPS-histogram: \(v.epe = v.eps + v.d\), using the histogram addition operation:

**Definition 25 (Histogram addition):** The addition of two time histograms \(H = H_1 + H_2\) is defined as

\[
H' = \{(p_{H_1}, t_{H_1}, p_{H_2}, t_{H_2}, p_{H_1} * p_{H_2}, t_{H_1} + t_{H_2}) | (p_{H_1}, t_{H_1}) \in H_1, (p_{H_2}, t_{H_2}) \in H_2\}
\]

\[
H = \{(\Sigma y, z) | (u, v, w, x, y, z) \in H'\}
\]

Propositions: The addition is commutative because \(H_1 + H_2 = H_2 + H_1\) and associative because \((H_1 + H_2) + H_3 = H_1 + (H_2 + H_3)\). A neutral element \(e = \{(1.0,0)\}\) representing a duration of 0 exists, such that \(H + e = e + H = H\). The addition can be extended to any number of histograms \(H = H_1 + H_2 + \ldots + H_n\) because of the operation's associativity.

The addition-operation creates the cartesian product of the tuples in the two histograms. The probabilities of each tuple-pair are multiplied and their time-values are added. The extended tuple notation in \(H'\) is needed to distinguish between multiple result-tuples with equal time and equal probability. Otherwise these tuples would be merged into one unique tuple, resulting in a time histogram with \(\Sigma p < 1\), which is no longer valid! The notation \((\Sigma y, z)\) ensures valid aggregation of tuples – it indicates that all tuples with equal time values of the intermediate set \(H'\) are aggregated into one tuple by adding their probability values. The resulting histogram does not hold values for each possible route,
but aggregated probability-information for every time value. Nodes of type activity, and-split, or-split, and end are always successors to a single dependency edge with a forward branching probability of 1. Assuming that there are no waiting times between nodes, their EPS-histograms are always equal to the EPE-histogram of their predecessors. The first few steps in the example are therefore:

\[
\begin{align*}
    \text{Start}.eps & = \{1.0, 0\} \\
    \text{A}.eps & = \text{Start}.eps + \text{Start}.d \\
                   & = \{1.0, 0\} + \{1.0, 0\} = \{1.0, 0\} \\
    \text{A}.epe & = \text{A}.eps + \text{A}.d \\
                   & = \{1.0, 0\} + \{(0.7, 3), (0.3, 5)\} \\
                   & = \{(0.7, 3), (0.3, 5)\} \\
    \text{B}.eps & = \text{A}.eps + \text{A}.d \\
                   & = \{1.0, 0\} + \{(0.7, 3), (0.3, 5)\} \\
                   & = \{(0.7, 3), (0.3, 5)\} \\
    \text{OS}.eps & = \text{B}.eps + \text{B}.d \\
                   & = \{(0.7, 3), (0.3, 5)\} + \{(0.5, 3), (0.5, 5)\} \\
                   & = \{(0.35, 6), (0.5, 8), (0.15, 10)\} \\
    \text{C}.eps & = \text{D}.eps = \text{OS}.epe = \text{OS}.eps + \text{OS}.d \\
                   & = \{(0.35, 6), (0.5, 8), (0.15, 10)\} + \{1.0, 0\} \\
                   & = \{(0.35, 6), (0.5, 8), (0.15, 10)\} \\
    \text{C}.epe & = \text{C}.eps + \text{D}.C \\
                   & = \{(0.35, 6), (0.5, 8), (0.15, 10)\} + \{1.0, 2\} \\
                   & = \{(0.35, 6), (0.5, 10), (0.15, 12)\} \\
    \text{D}.epe & = \text{C}.eps + \text{D}.d \\
                   & = \{(0.35, 6), (0.5, 8), (0.15, 10)\} + \{1.0, 4\} \\
                   & = \{(0.35, 10), (0.5, 12), (0.15, 14)\}
\end{align*}
\]

Consider the semantics of \text{B}.eps: B will start at 3 with a probability of 70% and at 5 with a probability of 30%. Now consider the histogram addition of \text{B}.epe = \text{B}.eps + \text{B}.d = \{(0.7, 3), (0.3, 5)\} + \{(0.5, 3), (0.5, 5)\}. The histogram addition results in four tuples \{(0.35, 6), (0.35, 8), (0.15, 8)\} and \{(0.15, 10)\}. According to the definition of the addition-operation tuples with equal time-values are merged. The resulting histogram contains aggregated probability-information for distinct time values: \text{B}.epe = \{(0.35, 6), (0.5, 8), (0.15, 8)\}.

4.5.1.2 Or-joins

The semantics of an or-join demands that exactly one route, out of multiple potential routes, can be executed, which is unknown at build time. The EPS-histogram of an or-join is calculated by weighting every predecessors EPE-histogram with the corresponding
branching probability, followed by a histogram disjunction to merge the weighted histograms.

**Definition 26 (Weight):** A time histogram $H_1$ is weighted by a probability $w$ resulting in $H_2$ as follows: $H_2 = H_1 \ast w = \{(p \ast w, t) \mid (p, t) \in H_1\}$. 

In the example $OJ$ has two in-coming edges: $C \Rightarrow OJ$ and $D \Rightarrow OJ$, with the forward branching probabilities $p_{C \Rightarrow OJ} = 0.3$ and $p_{D \Rightarrow OJ} = 0.7$. At first, the EPE-histograms of the predecessor nodes must be weighted: $(0.3 \ast C.epe)$ and $(0.7 \ast D.epe)$.

**Definition 27 (Histogram disjunction):** The disjunction $H = \bigvee_{i=1}^{n} H_i = H_1 \vee H_2 \cdots \vee H_n$ of $n$ histograms $H_1, H_2, \ldots, H_n$ is calculated as follows:

$$H' = \{(1, p_1, t_1) \mid (p_1, t_1) \in H_1\} \cup \{(2, p_2, t_2) \mid (p_2, t_2) \in H_2\} \cup \cdots \cup \{(n, p_n, t_n) \mid (p_n, t_n) \in H_n\}$$

$$H = \{((\Sigma v, w) \mid (u, v, w) \in H')\}$$

Proposition: If the unweighted input-histograms are valid and the sum of all join-probabilities is 1, then weight and disjunction always yield a valid and aggregated histogram.

Then the disjunction must be applied on the weighted histograms. Again $H'$ and $(\Sigma v, w)$ are necessary to guarantee that no tuple gets lost and that $H$ is aggregated. Now the or-join can be calculated, followed by some histogram additions:

$$OJ.epe = (p_{C \Rightarrow OJ} \ast D.epe) \vee (p_{D \Rightarrow OJ} \ast D.epe)$$
$$= (0.3 \ast C.epe) \vee (0.7 \ast D.epe)$$
$$= \{(0.105, 8), (0.395, 10), (0.395, 12), (0.105, 14)\}$$

$$AS.epe = OJ.epe = OJ.epe + OJ.d = OJ.epe$$
$$= \{(0.105, 8), (0.395, 10), (0.395, 12), (0.105, 14)\}$$

$$E.eps = F.eps = A_1.epe = A_1.epe + A_1.d = AS.epe$$
$$= \{(0.105, 8), (0.395, 10), (0.395, 12), (0.105, 14)\}$$

$$E.epe = E.eps + E.d$$
$$= \{(0.084, 11), (0.337, 13), (0.395, 15), (0.163, 17), (0.021, 19)\}$$

$$F.epe = F.eps + F.d$$
$$= \{(0.105, 12), (0.395, 14), (0.395, 16), (0.105, 18)\}$$

### 4.5.1.3 And-joins

After and-splits all succeeding routes will be processed concurrently which must be synchronized at and-joins. At the and-join $A_1$ time histograms of different routes must be
merged, and since the longest route must be considered a maximum conjunction must be applied.

**Definition 28 (Maximum conjunction):** The maximum conjunction $\land_{\text{max}}$ of two histograms $H_1$ and $H_2$ is calculated as

$$H = H_1 \land_{\text{max}} H_2$$

$$= \{ (\Sigma p, t) \mid (p_1, t_1) \in H_1, (p_2, t_2) \in H_2, t = \max(t_1, t_2), p = p_1 \ast p_2 \}$$

Propositions: If $H_1$ and $H_2$ are valid and aggregated, then $H$ will also be valid and aggregated. The operation is commutative because $H_1 \land_{\text{max}} H_2 = H_2 \land_{\text{max}} H_1$ and associative because $(H_1 \land_{\text{max}} H_2) \land_{\text{max}} H_3 = H_1 \land_{\text{max}} (H_2 \land_{\text{max}} H_3)$, thus the definition can be extended to $n$ histograms $H = H_1 \land_{\text{max}} H_2 \land_{\text{max}} \ldots \land_{\text{max}} H_n = \land_{\text{max}=1}^n H_i$.

For the max-conjunction every tuple $(p_1, t_1)$ of $H_1$ must be combined with every tuple $(p_2, t_2)$ of $H_2$. Whereby the maximum time value of $t_1$ and $t_2$ determines the time value of a tuple in $H$ and the product of the probabilities $p_1 \ast p_2$ determine the according probability value. Again $(\Sigma p, t)$ is used to guarantee that no tuple gets lost and that $H$ is aggregated. The resulting histogram holds time- and probability-information, considering a worst case view (as the longest route must be considered), on each possible combination of end times that stems from preceding parallel execution routes. In the example the EPS-histogram of $AJ$, followed by some histogram additions, is calculated as:

$$AJ.ep  =  E.ep \land_{\text{max}} F.ep
$$

$$=  \{(0.0088, 12), (0.0354, 13), (0.1663, 14), (0.1975, 15), (0.3223, 16), (0.1459, 17), (0.1028, 18), (0.021, 19)\}$$

$$End.ep  =  End.ep + End.d = AJ.ep = AJ.ep + AJ.d = AJ.ep
$$

$$=  \{(0.0088, 12), (0.0354, 13), (0.1663, 14), (0.1975, 15), (0.3223, 16), (0.1459, 17), (0.1028, 18), (0.021, 19)\}$$

Note that the EPE-histogram of the end-node represents the overall duration of the whole process.

### 4.5.2 Backward Calculation of L-histograms

To determine LAS and LAE-histograms of every node $v \in \mathcal{V}$ the backward calculations specified in table 4.3 have to be applied to each node in a backward topological order, according to the node type $v.t$, where $\text{succ} \in v.\text{Succ}$ is a successor node of $v$. The backward calculation is displayed in Fig. 4.13.
4.5.2.1 Adding the negated Duration

For the backward calculation time values must be subtracted from each other. Thus the LAS-histogram of each node $v \in V$ is determined by adding its negated duration-histogram to its LAE-histogram: $v.las = v.lae + v.d^-$. 

4.5.2.2 Sequence-like Calculation

In the backward calculation the node-types end, activity, and-join, and or-join are always predecessors to a single node. Therefore they must be treated like (reverse) sequences. As no waiting times between nodes are considered the LAE-histogram of such a node is always equal to its predecessor's LAS-histogram.

4.5.2.3 And-splits

To calculate LAE-histograms of an and-split a minimum conjunction must be applied to merge the LAS-histograms of all successors. The minimum conjunction considers the worst case of each tuple-combination which is determined by the minimum time value.
4.5 Calculation of the Probabilistic Timed Graph

Definition 29 (Minimum conjunction): The minimum conjunction of two histograms $H_1$ and $H_2$ is

\[ H = H_1 \land_{\min} H_2 = \{(p, \tau) \mid \exists (p_1, t_1) \in H_1, \exists (p_2, t_2) \in H_2, t = \min(t_1, t_2), p = p_1 \cdot p_2\} \]

Propositions: If $H_1$ and $H_2$ are valid and aggregated, then $H$ will also be valid and aggregated. The operation is commutative because $H_1 \land_{\min} H_2 = H_2 \land_{\min} H_1$ and associative because $(H_1 \land_{\min} H_2) \land_{\min} H_3 = H_1 \land_{\min} (H_2 \land_{\min} H_3)$, thus the definition can be extended to $n$ histograms $H = H_1 \land_{\min} H_2 \land_{\min} \ldots \land_{\min} H_n = \land_{\min_{\omega=1}}^{\omega} H_i$.

4.5.2.4 Or-splits

The LAE-histogram of an or-split is calculated by weighting the LAS-histograms of all successors with the corresponding backward branching probability, followed by a histogram conjunction (same operation as in the forward calculation) that aggregates the intermediate histograms.

For the running example the last node is initialized with $End.epe = \{(1.0, 19)\}$, where $\delta = 19$ denotes process deadline. To determine the L-histograms of all nodes the backward calculations specified above have to be applied as follows:

\[
\begin{align*}
E.lae &= F.lae = AJ.las = AJ.lae + AJ.d^- = End.las = End.lae + End.d^- \\
&= \{(1.0, 19)\} + \{(1.0, 0)\} = \{(1.0, 19)\} \\
E.las &= E.lae + E.d^- = \{(1.0, 19)\} + \{(0.8, -3), (0.2, -5)\} \\
&= \{(0.2, 14), (0.8, 16)\} \\
F.las &= F.lae + F.d^- = \{(1.0, 19)\} + \{(1.0, -4)\} = \{(1.0, 15)\} \\
AS.lae &= E.las \land_{\min} F.las = \{(0.2, 14), (0.8, 15)\} \\
C.lae &= D.lae = OJ.las = OJ.lae + OJ.d^- = AS.las = AS.lae + AS.d^- \\
&= \{(0.2, 14), (0.8, 15)\} + \{(1.0, 0)\} = \{(0.2, 14), (0.8, 15)\} \\
C.las &= C.lae + C.d^- \\
&= \{(0.2, 14), (0.8, 15)\} + \{(1.0, -2)\} = \{(0.2, 12), (0.8, 13)\} \\
D.las &= D.lae + D.d^- \\
&= \{(0.2, 14), (0.8, 15)\} + \{(1.0, -4)\} = \{(0.2, 10), (0.8, 11)\} \\
OS.lae &= 0.3 \cdot C.las \lor 0.7 \cdot D.las \\
&= \{(0.14, 10), (0.56, 11), (0.06, 12), (0.24, 13)\} \\
B.lae &= OS.las = OS.lae + OS.d^- \\
&= \{(0.14, 10), (0.56, 11), (0.06, 12), (0.24, 13)\} \\
A.lae &= B.las = B.lae + B.d^- \\
&= \{(0.14, 10), (0.56, 11), (0.06, 12), (0.24, 13)\} + \{(0.5, -3), (0.5, -5)\}
\end{align*}
\]
4 A Probabilistic Model for Acyclic Processes

\[
\begin{align*}
A.las &= A.lae + A.d^- \\
&= \{(0.07,5), (0.28,6), (0.1,7), (0.4,8), (0.03,9), (0.12,10)\} + \\
&\{(0.7,-3), (0.3,-5)\} \\
&= \{(0.021,0), (0.084,1), (0.079,2), (0.316,3), (0.079,4), (0.316,5), \\
&\quad (0.021,6), (0.084,7)\}
\end{align*}
\]

\[
\begin{align*}
Start.las &= Start.lae + Start.d^- = A.las \\
&= \{(0.021,0), (0.084,1), (0.079,2), (0.316,3), (0.079,4), (0.316,5), \\
&\quad (0.021,6), (0.084,7)\}
\end{align*}
\]

### 4.5.3 Backward Calculation of R-histograms

A variation of the backward calculation is used to determine remaining time histograms (R-histograms) \(v.rs\) and \(v.re\) for every node \(v \in \mathcal{V}\). An R-histogram contains possible remaining execution times (temporal distances from the node to the end of the workflow) and their corresponding probabilities. In the running example the last node's RE-histogram is initialized with \(End.re = \{(1.0,0)\}\). To determine the R-histograms of all remaining nodes the backward calculations specified in table 4.4 have to be applied depending on the node-type, yielding the following results:

\[
\begin{align*}
E.re &= F.re = A.f.rs = A.f.re + A.f.d = End.rs = End.re + End.d \\
&= \{(1.0,0)\} \\
E.rs &= E.re + E.d = \{(0.8,3), (0.2,5)\} \\
F.rs &= F.re + F.d = \{(1.0,4)\} \\
AS.re &= F.rs \land_{max} F.rs = \{(0.8,4), (0.2,5)\} \\
C.re &= D.re = OJ.rs = OJ.re + OJ.d = AS.rs = AS.re + AS.d \\
&= \{(0.8,4), (0.2,5)\} \\
C.rs &= C.re + C.d = \{(0.8,6), (0.2,7)\} \\
D.rs &= D.re + D.d = \{(0.8,8), (0.2,9)\} \\
OS.re &= 0.3 \cdot C.rs \lor 0.7 \cdot D.rs = \{(0.2400,6), (0.06,7), (0.56,8), (0.14,9)\} \\
B.re &= OS.rs = OS.re + OS.d \\
&= \{(0.2400,6), (0.06,7), (0.56,8), (0.14,9)\} \\
A.re &= B.rs = B.re + B.d \\
&= \{(0.12,9), (0.03,10), (0.4,11), (0.1,12), (0.28,13), (0.07,14)\} \\
A.rs &= A.rs + A.d \\
&= \{(0.084,12), (0.021,13), (0.316,14), (0.079,15), (0.316,16), \\
&\quad (0.079,17), (0.084,18), (0.021,19)\}
\end{align*}
\]
### Table 4.4: Backward Calculation Rules for R-histograms per Node Type

<table>
<thead>
<tr>
<th>v.t</th>
<th>(v.re =)</th>
<th>(v.rs =)</th>
</tr>
</thead>
<tbody>
<tr>
<td>end</td>
<td>{(1.0, 0)}</td>
<td>(v.re + v.d)</td>
</tr>
<tr>
<td>start</td>
<td>succ.rs</td>
<td>(v.re + v.d)</td>
</tr>
<tr>
<td>activity</td>
<td>succ.rs</td>
<td>(v.re + v.d)</td>
</tr>
<tr>
<td>and-split</td>
<td>(\land_{\text{max}}(\text{succ.rs}))</td>
<td>(v.re + v.d)</td>
</tr>
<tr>
<td>and-join</td>
<td>succ.rs</td>
<td>(v.re + v.d)</td>
</tr>
<tr>
<td>or-split</td>
<td>(\lor (\text{succ.las} \ast p_{V=\text{succ}}))</td>
<td>(v.re + v.d)</td>
</tr>
<tr>
<td>or-join</td>
<td>succ.rs</td>
<td>(v.re + v.d)</td>
</tr>
</tbody>
</table>

4.5 Calculation of the Probabilistic Timed Graph
Chapter 5

Interpretation and Application of Time Histograms

Time histograms, which are stored in the probabilistic timed graph, can be applied for build time as well as run time purposes. This chapter describes how time histograms are cumulated and interpreted, how to query them to receive forecasts about process durations, remaining times, and possible deadline violations, along with techniques to decrease the calculation and storage complexity of time histograms and timed graphs.

5.1 Cumulation and Interpretation of Time Histograms

After the calculation of the probabilistic timed graph, the question arises how to interpret and use all these different histograms.

5.1.1 Ascending Cumulated Time Histograms

Ascending cumulated time histograms can be used for time constraints and properties that stem from a temporal distance. This applies to D-histograms, E-histograms and R-histograms which describe a temporal distance between two nodes. As already mentioned, histograms are used to describe the discrete probability distribution function \( P_T(t) = P(T = t) \). For time management purposes it is very useful to present the likelihood of an outcome using the cumulated distribution function

\[
F_T(t) = P(T \leq t) = \sum_{T \leq t} P_T(t), t \in N.
\]

It describes the probability that the random variable \( T \) is less than or equal to a value \( t \). This is applied on cumulated time histograms in a straightforward manner.
Definition 30 (Ascending cumulated time histogram): An ascending cumulated time histogram $H$ is the transformation of a time histogram $H$ with tuples $(p, t)$ to a set of tuples $(c, t)$, with cumulated probability $c$ and time value $t$, such that

$$c_i = \sum_{t_j \leq t_i} p_j, \quad 1 \leq i \leq |H|.$$ 

Therefore cumulated time histograms are calculated from regular time histograms (or vice versa), as $c_i = p_1 + p_2 + \ldots + p_i$ (or $p_i = c_i - c_{i-1}$). Consider the running example: according to the results of the forward calculation, the original and cumulated EPE-histograms of $B$ are

$$B.epe = \{(0.35, 6), (0.5, 8), (0.15, 10)\}$$
$$1B.epe = \{(0.35, 6), (0.85, 8), (1.0, 10)\}$$

This cumulated histogram (see also Fig. 5.1) adheres to the following semantics: the probability that $B$ will be finished until 10 is 100%, for 8 it is 85% and for 6 it is 35%. Additionally, as $C.eps = D.eps = B.epe$, the same can be stated for the start of $C$ and $D$. The cumulated distribution function $F_T(t)$ is realized with the probability selection operation on an ascending cumulated time histogram. Prerequisites for the selection operations are two set-based operations.

- $\max\{v_1, \ldots, v_n\}$ which yields the maximum value of a set of values $v_1, \ldots, v_n$, and $\max(\emptyset) = 0$.

- $\min\{v_1, \ldots, v_n\}$ which yields the minimum value of a set of values $v_1, \ldots, v_n$, and $\min(\emptyset) = 0$.
Definition 31 (Ascending Probability Selection): The selection $^\sigma_x^p(H)$ of a probability value from a time histogram $H$ for a given time value $x$ is

$$^\sigma_x^p(H) = \max\{c \mid (c, t) \in H, t \leq x\}$$

The probability selection is used to express time management queries, like: "What is the probability that B will be finished until 9", stated as $^\sigma_9^p(B.epe) = 0.85$. In Fig. 5.1 the small circle indicates the selected tuple. If no element with the demanded time value exists, the cumulated probability of the next lower element must be selected.

Definition 32 (Descending Time selection): The selection $^\sigma_x^l(H)$ of a time value from a time histogram $H$ for a given probability $0 < x < 1$ is

$$^\sigma_x^l(H) = \min\{c \mid (c, t) \in H, c > x\}.$$  

The time selection is used to express another type of query, for instance $^\sigma_{0.90}^l(B.epe) = 10$ states that, with a probability of at least 90%, activity B will be finished by 10 (see also Fig. 5.1). If no element with the demanded cumulative probability exists, the time value of the next greater element must be selected. A time selection with $x = 100\%$ will always yield the maximal time-entry. Of course these operations can also be applied on EPS, D, or R-histograms. Consider the following examples:

- **Duration histograms**: $^\sigma_x^p(v,d) = y$ yields the probability $y$ that node $v$ will have a duration of $x$ or less. $^\sigma_y^p(v,d) = x$ means that, with a probability of at least $y$, the node $v$ will have a duration of $x$ or less. Consider the overall duration of the example process (which is equal to End.epe): $^\sigma_{15}^p(d) = 0.408$ yields the probability for a duration of at most 15. And $^\sigma_{0.90}^p(d) = 18$ means that the workflows duration will not be longer than 18 with a probability of at least 90%.

- **Remaining time histograms**: $^\sigma_x^p(v,re) = y$ yields the probability $y$ that, after the execution of node $v$, the workflows remaining duration will be $x$ or less. And $^\sigma_y^p(v,re) = x$ means that with a probability of at least $y$ the remaining time after the execution of $v$ will be $x$ or less.

5.1.2 Descending Cumulated Time Histograms

It has already been stated that, the greater the L-values in an L-histogram, the higher the probability of a deadline violation. This observation cannot be translated to cumulated distribution functions in a straightforward manner and therefore ascending cumulation cannot be applied.
5.1.2.1 Problem Statement

Consider the example presented in figure 5.2. The deadline is $\delta = 20$ and durations (D-histograms) are displayed on top of each activity. After the backward calculation the LAE-histogram of D is $D.lae = \{(0.75, 13), (0.25, 16)\}$. The probabilities for the l-values (abbreviated notation for LE-values) stem from possible distances of different routes between D and the end of the workflow. One can also see that the l-values and according distances $d$ can be calculated as $d = \delta - l$ or $l = \delta - d$ (see also Tab. 5.1). The values in this table can be interpreted in 2 ways

1. The distance between D and the deadline of 20 is 4, with a probability of 0.25. The distance between D and the deadline of 20 is 7, with a probability of 0.75

2. If D ends at 13 there is a probability of 0.75 that the workflow will end at exactly 20. If D ends at 16 there is probability of 0.25 that the workflow will end at exactly 20.

Thus an L-value is a point in time with respect to reach a certain deadline that corresponds to a distance between the activity and the activity on which the deadline is defined. The most important observations are: 1) probabilities used for l-values stem from distances, and 2) the greater the distance the lower the accompanying l-value. As ascending cumulation is performed in ascending order of time values, which implies that probabilities increase along with adhering time values, L-histograms must not be cumulated in an ascending order!

5.1.2.2 Proof

Assume that l-values are represented by the discrete random variable $L$, thus the random variable $D$ describing probabilities for distances can also be calculated as $D = \delta - L$, or
vice versa \( L = \delta - D \). From that follows that \( P_L(l) = P_D(\delta - d) \) and \( P_D(d) = P_L(\delta - l) \), for \( d = \delta - l \).

- \( P_D(d) \) specifies the probability that the distance to the end of the workflow equals some value \( d \)

- \( P_L(l) \) specifies the probability that a deadline can be reached (exactly) if D ends at \( l \)

A cumulated distribution function sums up probabilities in ascending order of values, thus it can be stated that greater values must yield higher probabilities. Applying the regular distribution function on L-histograms delivers incorrect results as the cumulated probabilities of L-values and relevant distances differ. Table 5.1 shows probabilities and cumulated probabilities for distances and l-values according to the cumulated distribution functions \( F_D(d) \) and \( F_L(l) \). From the above stated \( P_L(l) = P_D(\delta - d) \) must follow that \( F_L(l) = F_D(\delta - d) \) for \( d = \delta - l \). Assume that \( \delta = 20 \) and \( l = 14 \), thus the according distance must be \( d = \delta - l = 6 \). The cumulated distribution functions yield \( F_D(6) = 0.25 \) and \( F_L(14) = 0.75 \). This violates the equation \( F_l(l) = F_D(\delta - d) \).

5.1.2.3 Solution

The regular cumulated distribution function for distances is defined as \( F_D(d) = P(D \leq d) \) which must be equal to \( P(D \leq \delta - l) \) for \( d = \delta - l \). The condition can be transformed to \( P(\delta - D \geq l) \) which is equal to \( P(L \geq l) \), according to the above stated correlation between the random variables \( L = \delta - D \). Obviously \( P(L \geq l) \) is not equal to the original \( P(L \leq l) \). As probabilities of latest allowed end times must, in respect to a given deadline, correlate with probabilities of distances the cumulated distribution function for \( L \) must be defined as \( F'_L(l) = P(L \geq l) \). This reversed cumulated distribution function is represented as a descending cumulated time histogram.

**Definition 33 (Descending Cumulated Time Histogram):** A descending cumulated time histogram \(^1H\) is the transformation of a regular time histogram \( H \) with tuples \((p, t)\) to a set of tuples \((c, t)\), with descending cumulated probability \( c \) and time value \( t \), where \( c_i = \sum_{j \geq i} p_j \) where \( 1 \leq i \leq |H| \).

**Definition 34 (Descending Probability selection):** The selection \( \sigma_x^P(H) \) of a probability value from a time histogram \( H \) for a given time value \( x \) is:

\[
\sigma_x^P(H) = \min\{\{c \mid (c, t) \in ^1H, t \geq x\}\}.
\]

**Definition 35 (Descending Time selection):** The selection \( \sigma_x^T(H) \) of a time value from a time histogram \( ^1H \) for a given probability \( x \) is:

\[
\sigma_x^T(H) = \max\{\{t \mid (c, t) \in ^1H, c \leq x\}\}.
\]
The descending cumulated latest allowed end time histogram for activity $B$ of the original running example (see Fig. 4.11) is $^B.lae = \{(1.0,10), (0.86,11), (0.30,12), (0.24,13)\}$ (displayed in Fig. 5.3). For instance $^1\sigma^B_{0.30}(B.lae) = 0.30$ returns the probability for a workflow-execution without deadline violations if $B$ is finished until 12. If no element with the demanded time value exists, the cumulated probability of the element with the next higher time value is selected. And $^1\sigma^B_{0.75}(B.lae) = 11$ yields the maximum end time of $B$, such that no deadline violation will occur with a probability (certainty) of at least 75%. A time selection with 100% will always yield the minimal time-entry.

### 5.1.3 Slack Time Histograms

In a scalar model the slack time of a node is determined by the difference between LAE and EPE of a node: $X.se = X.lae - X.epe$. Figure 5.4 shows the cumulated versions of the following histograms:

$$X.epe = \{(0.5, 22), (0.3, 26), (0.2, 30)\}$$

$$X.lae = \{(0.4, 28), (0.2, 30), (0.4, 35)\}$$

The figure already indicates that problems may arise when executing the workflow because for some cases the latest allowed end time is less than the earliest possible end time. Since E and L-histograms adhere to different interpretation semantics the slack time histogram (S-histogram) must be applied to pull these different semantics together. The S-histogram is calculated by determining the probability for every possible time distance between each tuple of $X.epe$ and $X.lae$.

**Definition 36 (Slack Time Histogram):** The slack time histogram $v.et$ of a node $v$ with an earliest possible end time $v.epe$ and a latest allowed end time $v.lae$ is

$$H' = \{(p_1, t_1, t_2, p_2 * p_1, t_2 - t_1) \mid \}$$
5.1 Cumulation and Interpretation of Time Histograms

This operation is actually a variation of the regular histogram-addition. Note that it can analogously be applied to start times: SS-histograms are calculated with \( X.\text{eps} \) and \( X.\text{las} \). Due to the fact that slack time is actually a temporal distance, which must be cumulated in ascending order, the slack time histograms for the end of activity \( X \) are:

\[
\begin{align*}
X.se &= \{(0.08, -2), (0.04, 0), (0.12, 2), (0.06, 4), (0.08, 5), (0.2, 6), (0.1, 8), (0.12, 9), (0.2, 13)\} \\
\uparrow X.se &= \{(0.08, -2), (0.12, 0), (0.24, 2), (0.30, 4), (0.38, 5), (0.58, 6), (0.68, 8), (0.80, 9), (1.0, 13)\}
\end{align*}
\]

The probability selection can be used to find out if activities with negative slack exist: \( \sigma_{-1}^{P}(X.se) = 0.08 \), thus there is a chance of 8% that, after finishing \( X \), there will be negative slack. The time selection can for example be used to find the maximum and minimum slack-time availability at \( X \), determined as \( \sigma_{1.0}^{T}(X.se) = 13 \) and \( \sigma_{0.6}^{T}(X.se) = -2 \) respectively.

5.1.4 Histogram Compression

During the calculation of a probabilistic timed graph, it is very likely that huge duration histograms will be generated. Therefore compression operations must be introduced, which aims at keeping the number of entries in the histogram to a low level without losing too much information.
5.1.4.1 Compaction Operation

The compaction of a time histogram is achieved by successively removing tuples, such that at each removal the loss of information is minimal. The tuple that, if removed, changes the cumulated histogram least can be found by identifying the step of the cumulated histogram with the smallest area. As the semantics of ascending and descending cumulated histograms differ it is necessary to apply different version of compaction operations. Consider the following histogram presented on the left-hand side in Figure 5.5:

\[ N_{epe} = \{(0.048, 24), (0.48, 27), (0.528, 29), (0.96, 32), (0.964, 38), (1.0, 41)\} \]

\[ \text{compact} (N_{epe}, 3) = \{(0.48, 27), (0.96, 32), (1.0, 41)\} \]

It has been compacted to 3 tuples by applying Algorithm 5.1 – three areas had to be removed to reach the desired number of three entries in the time histogram. The compaction of a descending cumulated time histogram has to be adapted according to the mirrored step function (see Algorithm 5.2). Note that these algorithms are not optimized as their main intention is to clarify the basic concepts. To increase their performance tuples and

---

**Algorithm 5.1 Compaction of Ascending Cumulated Time Histograms**

Input: Maximum number of histogram entries \( m \)
Input: Histogram \( H \), where \((c_i, t_i)\) represent tuples at position \( 1 \leq i \leq |H| \) corresponding to the ascending sort order.

Output: The modified histogram \( H' \)

1. insert tuple \((c_0, t_0)\) into \( H' \), with \( c_0 := 0 \) and \( t_0 := t_1 \)
2. while \(|H'| > m\) do
3. \( \text{area}[i] := (c_i - c_{i-1} \ast (t_{i+1} - t_i)), \) where \( 0 < i < |H| - 1 \)
4. remove \((c_i, t_i)\) from \( H' \), where \( \text{area}[i] \) is minimal
5. end while
6. remove tuple \((c_0, t_0)\)
Algorithm 5.2 Compaction of Descending Cumulated Time Histograms

**Input:** Maximum number of histogram entries $m$

**Input:** $H$, where $(c_i, t_i)$ represent tuples at position $1 \leq i \leq |H|$ corresponding to the descending sort order.

**Output:** The modified histogram $H$

1: insert tuple $(c_0, t_0)$ into $\bar{H}$, with $c_0 := 0$ and $t_0 := t_1$
2: while $|\bar{H}| > m$ do
3: $area[i] := (c_i - c_{i-1} \ast (t_i - t_{i+1}))$, where $0 < i < |H| - 1$
4: remove $(c_i, t_i)$ from $\bar{H}$, where $area[i]$ is minimal
5: end while
6: remove tuple $(c_0, t_0)$

Areas may be stored in hash-sets, and only affected areas should be recalculated after each tuple-elimination (neighbors of eliminated tuples).

5.1.4.2 Cut Operation

Another compression techniques is based on the idea that entries below a specific threshold are not of interest for a workflow-scheduler and can therefore be dumped. The Algorithm 5.3) shows this procedure for ascending cumulated time histograms. Consider the example presented on the right-hand side Figure 5.5, where the threshold was set to 70%.

\[
\text{N.epe} = \{(0.048,24), (0.48,27), (0.528,29), (0.96,32), (0.964,38), (1.0,41)\}
\]

\[
\text{cut(N.epe,0.7)} = \{(0.528,29), (0.96,32), (0.964,38), (1.0,41)\}
\]

Since the histogram holds no tuple with $c = 0.7$, the tuple with the next-lower cumulated probability $(0.528,29)$ must be preserved. But the two steps (tuples) below can be eliminated. To cut a a descending cumulated time histogram the Algorithm 5.4) has to be applied analogously.

Algorithm 5.3 Cutting Ascending Cumulated Time Histograms

**Input:** Cut selector $cs$, defining the probability where to cut the histogram.

**Input:** Histogram $H$, where $(c_i, t_i)$ represent tuples at position $1 \leq i \leq |H|$ corresponding to the ascending sort order.

**Output:** The modified histogram $H$

1: find the tuple $(c_x, t_x)$, where $\overline{\sigma_x(D)} = t_x$
2: remove all tuples $(c_i, t_i)$, where $c_i < c_x$

5.2 Application at Build Time

Although the PT-Graph contains only relative temporal information, it can already be used to perform certain checks.
### Algorithm 5.4 Cutting Descending Cumulated Time Histograms

**Input:** Cut selector $cs$, defining the probability where to cut the histogram.

**Input:** Histogram $H$, where $(c_i, t_i)$ represent tuples at position $1 \leq i \leq |H|$ corresponding to the descending sort order.

**Output:** The modified histogram $H$

1. find the tuple $(c_x, t_x)$, where $\bar{\sigma}^D_{\text{cut}}(D) = t_x$
2. remove all tuples $(c_i, t_i)$, where $c_i < c_x$

---

### 5.2.1 Slack Times

To find out if slack is less than 0, which indicates that the deadline is too tight, apply a probability selection on the slack time histogram with the slack time value -1. If $\bar{\sigma}^P_{-1}(v.st) > 0$ for any $v \in V$, then the workflow designer should think about relaxing the deadline. Consider the following example:

$$\bar{X}.se = \{(0.08, -2), (0.12, 0), (0.24, 4), (0.30, 5), (0.38, 6),$$
$$ (0.68, 8), (0.80, 9), (1.0, 13)\}$$

The operation $\bar{\sigma}^P_{-1}(X.se) = 0.08$ indicates a chance of 8% for an upcoming deadline violation (detected at activity $X$). As not every node in a workflow will be executed with the same probability, it is necessary to include the execution probability of $X$: $X.x \ast \bar{\sigma}^P_{-1}(v.st)$.

### 5.2.2 Satisfiability of a Probabilistic Timed Graph

The satisfiability is a percentage, specifying the probability for a valid workflow execution, based on the slack time and execution probability of all nodes in the graph.

**Definition 37 (Satisfiability):**

$$s = 1 - \max\{ p \mid p = v.x \ast \bar{\sigma}^P_{-1}(v.st), v \in V \}$$

The expression calculates the probability of negative slack for every node and weights this probability with the corresponding execution probability. One minus the highest probability of negative slack yields the minimum probability for an execution without constraint violations, called satisfiability. For graphs with $s < 1.0$ the designer should relax the deadline or redesign the graph to guarantee a violation-free execution. Alternatively, the designer could also initiate the optimization of activities, especially those with outliers in the upper histogram regions of their durations. The designer may also decide to accept a risk, leave the current graph unchanged, and accept for instance $s > 0.95$ which implies that presumably 5% of all instances will violate the deadline.
5.2.3 Identification of Critical Activities and Critical Paths

Critical activities are activities, that, if delayed, will presumably cause a deadline violation. The critical path is a sequence of nodes, where at least one of them is critical. In a probabilistic model an activity is critical, if $\sigma_P(v, st) > 0.0$. Again the designer might want to take the risk and accept probabilities below a certain threshold. During build time this information can for instance be used in a graph-based design tool mark critical activities (and paths). It helps the designer by identifying critical process regions and provides the means of optimizing them, which will eventually result in less deadline violations during run time.

5.3 Application at Process Instantiation

5.3.1 Adjusting The Deadline

In case that deadlines are changed at process instantiation the graph’s L-histograms must be recalculated, depending on the type of the deadline, as follows:

1. If the original process deadline $\delta$ is defined by a scalar value, as well as the new deadline $\delta'$, then the L-histograms can be adjusted quite easily by applying an add-scalar operation on each node $v \in V$: $v.lae' = v.lae + (\delta' - \delta)$ (same for LAS). Additionally it is necessary to adjust S-histograms by the same value: $v.se = v.se + (\delta' - \delta)$ (analogously for $v.ss$).

2. The first approach can not be applied if the deadline is defined by means of a histograms. As there exists no histogram-subtraction operation, it is necessary to repeat the backward calculation for L-histograms, with an initial $End.lae = \delta'$, followed by a recalculation of all S-histograms.

5.3.2 Calendar Mapping

At process instantiation it is also necessary to replace every relative point in time in the PT-Graph with absolute points in time. This is achieved by calendar mapping, which (in its simplest form) means adding the current system time $now$ to every E and L-histogram, e.g. $E.epc_{cal} = E.epc + now$, $E.lac_{cal} = E.lac + now$, etc.

5.4 Predictive Time Management at Run Time

During run time the time management component monitors the temporal status of all currently active process instances. According to the ideas in predictive time management, participants are provided with predictions about the future temporal status as well as the likelihood of events.
5.4.1 Prediction of Process End and Remaining Duration

Prediction includes forecasts about the remaining process duration (based on R-histograms) and activity durations (based on D-histograms). As it does not make much sense to present histograms to users, it will be necessary to select only one time value by applying a time selection. This requires the configuration of a probability which depicts the selection-parameter for the operation. It is advisable to pre-configure this value for each process, or even for each activity. The histogram on the left hand side of Figure 5.6 shows how to apply a time selection with 90% on an R-histogram: \( \sigma_{0.9}(X.re) = 19 \) cuts the upper 10% (eliminates outliers) and can be interpreted as “the process will be finished before now + 19 (with a probability of at least 90%)”. The time selection on D-histograms, which yields the expected remaining time of the according activities, works analogously. In processes with parallel structures multiple activities will be active at the same time. In this case the process remaining duration is determined by the maximum rest time value, selected from each of the R-histograms.

![Figure 5.6: Querying an R-histogram and Adjusting a D-histogram](image)

5.4.2 Histogram Adjustment

For D, R, and E-histograms adjustments are required if the corresponding activity has already started. The right hand side of Figure 5.6 shows the adjustment of a duration-histogram; activity X has been started at start\(X\) and the current time is depicted by now: \( X.d_{adj} = cut_{0}((X.d - (start_{X} - now)) \). The adjustment actually consists of two parts: at first the D-histogram is reduced by the node’s execution time until now, denoted by the scalar value start\(X\) – now, followed by a cut\(_{0}\)-operation on the resulting histogram.
Definition 38 (The \textit{cut} \textsubscript{0} \text{Operation}): Given that \( H, H' \) and \( H'' \) are time histograms, \( H'' = \text{cut}_{0}(H) \), such that

\[
\begin{align*}
H' &= \{ (p,t) \mid (p,t) \in H, t > 0 \} \\
H'' &= H' \cup \{(1 - \Sigma p, 0)\}, \text{ where } (p,t) \in H'
\end{align*}
\]

This operation performs a vertical cut through the histogram, by eliminating all tuples with a time value less or equal than zero. To ensure the validity of the histogram (sum of probabilities must be 1), a tuple \((1 - \Sigma p, 0)\) must be inserted.

5.4.3 Prediction of Deadline Violations

The PT-Graph enables the implementation of simple but effective escalation warning mechanisms, using an adaption of the 	extit{traffic light model} (introduced in Section 3.6.3.2), which is primarily based on L-histograms. Consider the following example: assume that activity \( X \) just finished, at a point in time denoted by \( \text{now} = 12 \). Figure 5.7 shows \( X.\text{lae}_\text{cal} \) with two thresholds, specified at 90\% and 50\% (the subscripted 'cal' indicates that the histogram has already been calendar mapped). The probability selection yields \( \gamma_{\text{12}}^{p}(X.\text{lae}_\text{cal}) = 0.30 \), which is the probability that the process will be finished without violating the overall deadline. This switches the status to red. In workflows with parallel structures multiple activities will be active at the same time. In this case the temporal state is determined by the minimum probability, selected from each of the L-histograms. According to the new state different escalation actions can be invoked (as described in Section 3.6.4). The important contribution of this approach is that threshold values can be expressed as the probability of a deadline violation.

The upper part of Figure 5.8 shows how the traffic light model can be integrated to the worklist of a participant. It is assumed that each workflow participant is responsible for executing activity instances from different process instances. The worklist contains a due-due for the completion of a task, which has been selected from the corresponding \( \text{LAE-} \)
histogram, for instance with a selection-probability specified by the threshold-probability between 'green' and 'orange' – note that the higher this threshold-probability, the earlier the due-date will be. The temporal status additionally indicates how late the process is – in case of a 'red' or 'orange' status the participant is warned and will try to catch up.

5.4.4 Prediction of Upcoming Activities

Another application is the prediction of upcoming activities, based on the E-histograms of activities which succeed the current activity. This allows workflow participants to improve their performance as they are no longer surprised by new entries in their work-lists. They receive information about (potential) future activity assignments, which allows work-planning in advance. The lower part of Figure 5.8 shows the integration of forecasts into the worklist. Note that these activity instances are possible future-tasks to be assigned to the participant. The expected arrival time is determined by the minimum time value in the corresponding EPS-histogram. The percentage is the probability of assignment (execution probability) – some may not be assigned at all due to branching decisions that lead to different execution-routes in the process. According to this forecast the participant knows that several tasks will (presumably) arrive on 26.10.2006 – determined by a time-selection on the corresponding EPS-histogram. The selection-probability must, similar to the traffic-light, be specified in advance. It will not always unambiguously determinable

<table>
<thead>
<tr>
<th>ID</th>
<th>Process</th>
<th>Task</th>
<th>Received</th>
<th>Due</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Order5</td>
<td>accounting</td>
<td>01.10.2006</td>
<td>23.10.2006</td>
<td>red</td>
</tr>
<tr>
<td>134</td>
<td>Order10</td>
<td>check offer</td>
<td>05.10.2006</td>
<td>27.10.2006</td>
<td>orange</td>
</tr>
<tr>
<td>256</td>
<td>Claim17</td>
<td>review</td>
<td>18.10.2006</td>
<td>30.10.2006</td>
<td>green</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Process</th>
<th>Task</th>
<th>Expect [est.]</th>
<th>Due [est.]</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>345</td>
<td>Order10</td>
<td>accounting</td>
<td>26.10.2006</td>
<td>1.11.2006</td>
<td>100%</td>
</tr>
<tr>
<td>349</td>
<td>Claim22</td>
<td>review</td>
<td>26.10.2006</td>
<td>2.11.2006</td>
<td>75%</td>
</tr>
<tr>
<td>367</td>
<td>Order26</td>
<td>check offer</td>
<td>26.10.2006</td>
<td>3.11.2006</td>
<td>25%</td>
</tr>
</tbody>
</table>

Figure 5.8: Worklist with Forecasts about Upcoming Activity Assignments

which participant is going to execute a specific future activity – for instance in a role-based system. In that case additional information must be added to the forecast. E.g. the role which will receive the activity, or if the assignment is not yet certain the percentage must be reduced accordingly.
In [37] we described how to apply probabilistic time management for personal-scheduling purposes. Based on timed graphs one can calculate the future workloads for each participant, represented as a schedule where upcoming activity instances are planned according to their valid execution intervals. This schedule can also be used to identify overloaded periods and to optimize them, by shifting activities within their valid execution intervals. Note that personal schedules are not comparable with machine scheduling plans, because a participant still has the freedom of choice whether she executes a certain activity or not, and when she executes it. For further details please refer to [37].

5.5 Updating the PT-Graph

In Section 3.6.3.4 it has been laid out why it is necessary to recalculate the PT-Graph periodically in order to adjust E-histograms to the current temporal state. The following describes how this is achieved in a PT-Graph.

5.5.1 Updating the PT-Graph at Run Time

If at certain events, like the start of an activity, the current system time now considerably differs from the EPS-histogram a partial recalculation of the PT-Graph, starting from the current activity, is recommended. Otherwise, especially E-histograms will become virtually useless, as forecasts based upon them will greatly vary from the actual execution. Figure 5.9 shows a situation where activity X has just been finished. The current time now indicates that predecessor activities have been executed faster than expected. Therefore the actual end of X happened much earlier than indicated by the build time-calculated earliest possible end. This implies that forecasts about assignments of upcoming activities will be wrong; their actual assignment will take place earlier than expected. Thus it is recommended to update the E-histograms of the PT-graph if now is outside the scope of the according E-histogram. Fig. 5.10 shows a scenario with parallel structures. The tokens indicate the current state of execution. The execution probabilities are already adapted to

![Figure 5.9: Faster than Expected](image-url)
this current state – as A, C and D are currently active, their execution probability is 1. Execution probabilities, forward and backward branching probabilities, and finally, already calendar mapped E-histograms are mostly calculated as already described, in consideration of the following differences: 1) the predecessors of A, C and D are not considered at all, 2) A, C and D are treated like start nodes, 3) they are initialized with $ep_{cal} = \{(1.0, now)\}$, and 4) their D-histograms must be adjusted to the current time now, as explained above.

It is not necessary to recalculate L-histograms – their values won’t change since their histograms are influenced by successors, originating from the deadline at the end of the process. If one insists on calculating these values take care of this: LAE-histogram must be initialized with the already calendar mapped deadline $lae = \{(1.0, start_{Start} + \delta)\}$, not $now + \delta$, as this deadline still refers to the start of the process. Regarding calculation calendar mapped histograms: it makes perfect sense to calculate an already calendar mapped version of the graph, as this version will be valid for the current process instance only!

### 5.5.2 Splitting the PT-Graph at Build Time

Updating a PT-Graph during process execution is time and resource consuming, and moreover, the already calendar mapped updated graph can only be used for the current process instance. Therefore it is advisable to split a PT-Graph in several parts and calculate information for these parts in isolation at build time. To do this one must find suitable synchronization points – also called sync-points – in the graph. At these points the temporal model can be synchronized with the current system time during run time.

#### 5.5.2.1 Identification of Sync-Points

Figure 5.11 shows an example process and indicates possible sync-points. A sync-point must have an execution probability of 100%, because every process instance will pass such a point during execution. But beware: as sync-points must adhere to one specific run time state of a process, nodes inside a parallel structure, even those with an execution
probability of 100%, are ill-suited candidates. Consider the following example: activity C has an execution probability of 100%, which means that it will always be executed during run time. But as C resides in a parallel structure one does not know where the other token will be during the execution of C; it could be at A, B, or at one of the control nodes. As join-nodes synchronize the flow of execution they are extremely well suited. Therefore join nodes or their direct successors which are not nested in other structures and have an execution probability of 100% are sync-point candidates. For the graph in Fig. 5.11 the activities F and E were chosen as sync-points.

5.5.2.2 Time Calculation Issues

After the identification of sync-points it is necessary to split the graph in several parts. The graph in Fig. 5.11 can be split in three parts: the first part from Start to OJ1, the second one from F to AJ1, and the third one from E to End. These three parts can be treated and calculated similar to regular PT-graphs. Still the following issues must be considered:

- **Forward Calculation** Each sync-point acts as start node, initialized with an epe = \((1.0,0)\). Forward calculations for partial graphs are conducted in isolation. When passing a sync-point during run time, the time manager must automatically switch to the next partial graph, and the E-histograms must be calendar mapped by adding \(start_{\text{Start}}\). For build time reasoning purposes it is necessary to add the process durations of prior partial graphs, constituted by their last node’s EPE-histogram, to each E-histogram in a partial graph.

- **Backward Calculation** The backward calculation starts, as usual, by initializing the LAE of the end-node with the process deadline. Then all L-histograms of the last partial graph must be calculated. The LAS-histogram of the last sync-point constitutes the deadline for the last but one partial graph, and so on. For L-histograms no further run time adaptations, except calendar mapping, are necessary.

Splitting a process graph has an additional advantage: the complexity will be reduced enormously, which comes in extremely handy when unfolding arbitrary cycles (see Section 6.6).
5.6 Proactive Time Management at Run Time

The following describes two proactive probabilistic strategies which are applied in the simulation scenarios described in Chapter 7.

5.6.1 Probabilistic Process Prioritization

Section 3.6.4.3 already discussed several process prioritization strategies to be applied on the client side. The following new strategies utilize probabilistic information stored in the probabilistic timed graph.

- **Most Probable Deadline Violation - MPDV** This strategy aims at keeping the number of deadline violations as low as possible. It sorts the worklist according to deadline violation probabilities, according to the traffic light model which is based on LAE-histograms (see Section 5.4.3). The more likely a future deadline violation for the process instance of a specific work item is, the higher it will be ranked in the worklist. Therefore already late process instances will be favored over timely ones.

- **Lowest Proportional Slack - LPS** This strategy aims at reducing the tardiness of processes. It sorts the worklist according to available *proportional slack*. Proportional slack sets the available slack in relation to the remaining time. This makes sense as for example a slack time of 5 in a workflow instance with a remaining duration of 10 is a much greater buffer than for a workflow instance with a remaining duration of 100. The slack time can be determined as described in Section 5.1.3, by applying the probability-selection on the R-histogram. The same applies for the remaining time. Now the proportional slack can be calculated for each work item as \( ps = \frac{\text{slack}}{\text{remaining time}} \). It is used to sort the worklist in ascending order. Note that for this strategy a selection-probability must be specified. In subsequent chapters the notation LPS-y to indicate the usage of the LPS-strategy with a selection-probability of y% is applied.

5.6.2 Probabilistic Early Escalation

This strategy operates on the server side on a process instance level. It uses a probabilistic abort strategy. The concept stems from the (non-probabilistic) early-escalation approach described in Section 3.6.4.2. The basic idea is to abort process instances where the likelihood of a deadline violation in the future is very high. Hopelessly late instances block valuable resources without a chance of reaching their deadline. Early termination of these instances frees resources which speeds up still 'saveable' instances. The probability of not violating a deadline is determined by means of the traffic light model. Additionally a termination-probability \( x \) must be specified which indicates that process instances with a less-than-x probability of meeting the deadline shall be aborted immediately.
Chapter 6

Probabilistic Time Management for Cyclic Processes

Until now only acyclic structures have been discussed, but when modelling real business processes the usage of cyclic structures is indispensable. This chapter introduces necessary extensions to the probabilistic timed graph: an unfold-algorithm along with the adjustments of calculation rules for implicit time constraints.

6.1 Introduction

For the calculation of a timed graph the existence of cyclic structures generates severe problems, as time properties and implicit constraint are tremendously influenced by the number of loop iterations which must be resolved somehow. Although some (partial) solutions for the treatment of cyclic structures exist, they are inhibited by several major drawbacks.

6.1.1 Problem Statement

Similar to or-splits, the decision to enter another loop, is made at run time, and therefore not foreseeable. Furthermore prior chapters assumed that each activity generates exactly one activity instance during the execution of a process instance, which does no longer hold for activities in cyclic structures since they may be executed multiple times. This also implies that each activity instance ought to have its own valid execution interval, one for each iteration. Additionally, the existence of loops will have a large effect on the actual process duration, remaining times, and implicit time constraints, which will vary considerably depending on the actual number of iterations.

6.1.2 Existing Methods of Resolution

Existing time management literature quite frequently ignores the problem with cyclic structures (e.g. [58, 120, 122, 72]). However, some solutions have been proposed:
• **Complex Activity** The most frequently applied technique is to encapsulate the whole cycle in a complex activity with an average duration (e.g. [34, 78, 6]). This, naturally, deprives one of the possibility of reasoning about temporal properties of activities nested inside this structure – it must be treated as a black box.

• **Roll-out Sequential** The cyclic structure is transformed into a sequence, based on a given (expected, average, maximum) number of iterations (e.g. [88, 84, 98]). This technique provides the means to calculate time-properties for each activity instantiation inside the cyclic structure. For instance worst-case estimations (maximum number of iterations) make sense in a constraint-satisfaction scenario as they only have to consider the path with the longest duration at build time. Unfortunately it is not applicable for run time purposes, as the actual number of iterations will vary from the estimated ones, resulting in predicted time values that will be far-off the actual ones.

• **Roll-out Conditional** This technique proposes to transform the process to an acyclic structure which consists of conditional building blocks that are stringed together. Each iteration – a duplication of the inner structure – is followed by an or-split that either jumps out of the loop or enters another iteration. The branching probabilities for or-splits are determined from 'looping probabilities' that vary with each iteration (e.g. [36, 111]).

### 6.1.3 Some Problems Remain

All existing roll-out techniques are based on the assumption that all loops are block-structured. This renders them useless for workflow systems which allow for arbitrary cycles. Furthermore they do not consider variations of activity instance durations – in many cases the duration of an activity nested in a loop will vary with each instantiation. Often it will decrease with each anew iteration, e.g. in a form-based review process where data will be entered or corrected until it is valid.

### 6.2 Related Work

The table 6.1 extends the overview of existing time management approaches presented in Section 4.2. The table compares structural support including cyclic structures. Again Jasper/Zukunft and Bussler are not considered, as the corresponding publications do not provide enough details for a comparison. The table differentiates between blocked and arbitrary cycles. Approaches, which simply suggest or propose – without describing any details – to hide a blocked cycle in a complex activity or to roll it out to a sequence received a - (minus) in the corresponding row, as this can be easily realized in any approach. Approaches that actually describe a roll-out-sequential procedure received a -/+ , and approaches that describe a roll-out-conditional procedure received a +.
As already stated, many, even newer, approaches simply ignore the problem (e.g. Jasper and Zukunft [58], Zhao and Stohr [120], Zhuge et al. [122], Lu and Sadiq et al. [72]). Others propose encapsulating the cycle in a complex activity with an average duration (e.g. Eder and Panagos et al. [34], Marjanovic and Orlowska et al. [78], Bettini et al. [6]).

As a matter of fact only few authors propose rolling out or unfolding cyclic structures. Some (e.g. Eder et al. [88], Panagos and Rabinovich [84]) dispose it as lapidary problem, only mentioned in a subordinate clause, to be solved by applying an average number of iterations – however, details remain unclear. Son and Kim et al. [98] show how to transform a cycle into to a sequential structure, solely applicable for their critical-path approach, as they only have to consider the path with the longest duration. Eder and Pichler [36] and van der Aalst et al. [111] propose transforming the cycle into a conditional structure.

Roll-out techniques have one thing in common – they use a probabilistic representation to describe the iterative behavior. The approach described by Son and Kim et al. [98] is based on queuing theory – it views a cyclic structure as independent queuing system whose arrival rates can be described by the means of a Poisson process, including feedback caused by the iterative behavior. However they solely use it to roll the cycle out to a sequential structure, as already mentioned above. Eder and Pichler [36] introduced arbitrary discrete distributions to describe the branching behavior of every iteration, as the
probability of executing another loop will vary, often decrease, depending on the current iteration. Shortly afterwards a similar idea was proposed by van der Aalst et al. [111], which describes the iterative behavior of a cyclic block as a Bernoulli-distributed random variable, stating that each new application of this block is accompanied by the application of a new independent random variable. The idea itself is not new, it can already be found in process simulation tools, like Adonis from BOC\(^1\), which allows the definition of branching probabilities, according to the instantiation of an activity.

The duplication of join-nodes and differentiation of instance types, applied in this chapter, is inspired by the unfold-technique described by Eder et al. [29, 30] and Gruber [49]; although their approach aims primarily at the evaluation of constraint-satisfiability and does not consider cyclic structures.

### 6.3 Basic Concepts and Definitions

Unfolding of conditional structures – regardless of whether they are acyclic (cp. [30, 49]), blocked cyclic, or arbitrary cyclic – poses several problems for the calculation and interpretation of the timed graph. The concept itself is best explained by means of a simple example, as presented in Fig. 6.1. It shows the original graph of a simple cyclic workflow augmented with (for introductory reasons) scalar expected durations and branching probabilities. As already mentioned a loop can be modelled without introducing specific control-flow elements, just by using or-splits. Figure 6.2 shows the unfolded (rolled-out) version of this graph – note that instantiations of OJ have been omitted to save some space. It shows possible execution paths for nodes and their corresponding execution probabilities, along with tables that contain timing information for each activity. Additionally assume that the workflow deadline is \( \delta = 20 \). This rolled-out version of the graph has, due to its original cyclic nature, several peculiarities:

- **Hits** As activities A and B are encapsuled in a loop, they may be executed multiple times – therefore each instantiation of an activity is called hit. The hit-counter distin-

\(^1\)http://www.boc-eu.com
guishes between each instantiation of a node. E.g. the first execution of B, referred to as hitA, will be denoted B1, followed by B2, B3, and so on. Additionally, although not considered in this example, the duration of different hits may vary considerably.

- **Hit-versions** Additionally the unfolded graph may also contain multiple versions of one hit. E.g. consider C: as it does not reside inside the loop, it can only be executed once. But each ‘version’ of C1 resides on a different execution path (instance type), which means that each of them will have differing execution intervals. Therefore it is necessary to distinguish different versions of one hit, called *hit-versions*, by referring to them as follows: C1, C2, and so on. For nodes with exactly one hit-version, e.g. the first hit of A, still will be used as a shortcut.

- **Execution Probabilities** As there is a cyclic branch in the graph, the execution probability of different hit-versions will be different.

- **Multiple E and L-values** As each node may be executed more than once it is necessary to distinguish the E-values of different hits, e.g. B1.eps = 2, B2.eps = 8, B3.eps = 14, B4.eps = 20, and so on. Note that B4.eps already violates the deadline δ = 20. Furthermore it is obvious that this unfolded graph is potentially infinite which is problematic for the backward calculation of L-values.

### 6.3.1 Extended Workflow Graph for Cyclic Structures

The new extensions aim at process structures that contain arbitrary cycles. Please note that models that allow arbitrary cycles do implicitly allow the specification of blocked cycles as well. For a probabilistic technique that solely aims at blocked cycles please refer to [36].

#### 6.3.1.1 Par-blocked - a new Conformance Class

The definition language of the workflow system @enterprise, which supports the goto-statement, allows the definition of process models that contain arbitrary cycles. As we did not succeed in coming up with a solution for totally unconstrained non-blocked structures, it was indispensable to constrain them somehow, without narrowing the modelling
capabilities of the system. After a thorough examination of allowed and prohibited process structures along with the executional semantics offered by Enterprise, it was finally possible to come up with a new conformance class that allows a solution for arbitrary cycles.

**Definition 39 (Par-blocked):** A par-blocked workflow model has exactly one entry-node and exactly one exit-node. A parallel structure must be modelled as block, further on called *par-block*, which starts with an and-split and finishes with an according and-join. Each branch of a parallel structure, further on called *par-branch*, must itself be par-blocked.

Figure 6.3 shows a par-blocked and a non-par-blocked workflow. As one can see par-blocked allows for instance backward transitions, arbitrary loops, interleaving loops and nested par-blocks. Not allowed are transitions into or out of par-blocks or par-branches. Accordingly every par-blocked workflow must be sound, but not every sound workflow must be par-blocked. For instance in a sound workflow an and-split is not necessarily linked to one specific and-join. The conformance classes are in the following inclusion relation: non-blocked \( \supset \) par-blocked \( \supset \) full-blocked. As for loop-blocked it can be stated that workflow structures \( s \) exist, such that \( s \in \text{par-blocked} \cap \text{loop-blocked} \), where \( s \neq \emptyset \). Obviously this conformance class constrains the degree of modelling freedom a bit, but as stated in [65], jumping into or out of parallel structures is one of the main problems in process design, as this often violates the soundness criteria and results in control flow errors during run time. Therefore its usage is prohibited in many systems.

### 6.3.1.2 Basic Assumptions

The probabilistic unfold technique is based on the following assumptions which hold for the vast majority of (administrative and production) workflows:

1. The duration of a workflow is usually finite, this implies a finite number of executed nodes, and it implies furthermore that the number of loop-iterations will also be finite.
2. The probability that a loop will be executed another time tends to get smaller at each iteration. The path that exits the loop will usually be chosen after a small number of iterations.

3. Branching probabilities of regular or-splits, nested inside a cyclic structure, varies with each iteration of the loop.

4. The duration of an activity that is nested in a loop varies with each hit. Often it will decrease with each iteration, as for instance in a form-based review process where data will be entered or corrected until it is valid.

6.3.1.3 Definitions

To capture these assumptions the nodes of the probabilistic extended workflow graph model must be extended with branching and duration information for each node on a per-hit basis, which can – similar to D-histograms – be extracted from the workflow log.

Definition 40 (PE-Node for Cyclic Graphs): A node \( n.t \in V \) of a probabilistic extended workflow graph with cyclic structures, represents an activity or control node, where \( n.t \in T \) depicts its type, \( n.DT \) depicts its duration table and \( n.BT \) describes its branching table.

<table>
<thead>
<tr>
<th>Successors</th>
<th>#</th>
<th>A</th>
<th>C</th>
<th>SUM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>0.25</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>0.15</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>0.05</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>0.01</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
<td>0.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant Duration</th>
<th>A.DT</th>
<th>#</th>
<th>d</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differing Durations</td>
<td>D.DT</td>
<td>#</td>
<td>d</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6.4: Branching and Duration Tables**

Definition 41 (Branching Table): The attribute \( n.BT \) describes the branching table \( BT \) for a node \( n \in V \) as set of tuples \( (h, B_h) \). \( B_h \) defines the branching behavior of the hit \( n^h \). \( BT \) contains \( H + 1 \) entries such that \( B = \{(1, B_1), \ldots, (H, B_H), (\infty, B_\infty)\} \). For a node with \( M = |n.Succ| \) successors, \( B_h \) defines the branching behavior for hit \( h \) as \( B_h = \{(s_1, b_1), \ldots, (s_M, b_M)\} \), where \( 1 \leq m \leq M \) depicts the branching probability \( b_m \) to a successor \( s_m \in n.Succ \). The additional tuple \( (\infty, B_\infty) \) describes the branching behavior for hits above the contained maximum hit \( H \). The expression \( BT.getProbability(h, s) \), where \( h > 0 \), yields the probability \( x \) if \( (h, B_h) \in B \land (s, x) \in B_h \), the probability \( y \) if \( (h, B_h) \notin B \land (\infty, B_\infty) \in B \land (s, y) \in B_\infty \).
The branching table in Figure 6.4 represents the branching behavior of the or-split OS on a per-hit basis:

\[
OS.BT = \{(1, \{(A, 0.25), (C, 0.75)\}), (2, \{(A, 0.15), (C, 0.85)\}), (3, \{(A, 0.05), (C, 0.95)\}), (4, \{(A, 0.01), (C, 0.99)\}), (\infty, \{(A, 0.00), (C, 1.00)\})\}
\]

As one can see after hit\#4 the successor of OS will always be activity C. For the sake of universality branching tables are stored for every node, even for those with only one successor – for unconditional branching with a probability of 100%. This representation allows also loops with a fixed number of iterations (e.g. for-next).

**Definition 42 (Duration Table):** A duration table \(v.DT\) for a node \(n \in V\) is a set of tuples \((h, d)\), where \(d\) depicts the duration for hit \(h\) of the node \(n\). \(DT\) may contain \(H\) entries such that \(DT = \{(1, d_1), \ldots, (H, d_H), (\infty, d_\infty)\}\). The additional tuple \((\infty, d)\) describes the duration for hits above the maximum \(H\) in \(DT\). The expression \(DT.getDuration(h)\), where \(h > 0\), yields the duration \(x\) if \((h, x) \in DT\), or the duration \(d_\infty\) if \((h, x) \notin DT\).

The first duration-table example in Figure 6.4 represents a constant duration of activity A, which is equal for each hit.

\[
A.DT = \{(\infty, 2)\}
\]

As for D, the durations will vary, from hit\#1 to hit\#3, and for all hits after hit\#3 the duration will be 1:

\[
D.DT = \{(1, 45), (2, 7), (3, 5), (\infty, 1)\}
\]

**6.3.1.4 Further Considerations**

It is quite obvious that a hit-based representation does not allow distinguishing between the duration and branching-behavior of different hit-versions. The reasons for that are very similar to the reasons explained in Section 4.3.4: it would be necessary to store instance-type-specific information which would complicate the representation model considerably.

The independency-assumption also implies that the branching-behavior of an outer-loop does not influence the branching behavior of its inner-loop, and vice-versa.

Please note that duration tables can easily be extended to hold D-histograms instead of simple scalar values. Only for the sake of simplicity of subsequent explanations and examples a scalar representation of duration values will be used throughout this chapter.
6.4 Unfolding Arbitrary Cycles

The objective of this technique is to transform the cyclic extended workflow graph to an acyclic timed graph — called unfolded graph — which can then be used for the calculation of the timed workflow graph along with all implicit time constraints and time properties. Applying branching probabilities from branching tables, instead of constant branching probabilities, during the unfold-procedure limits the size of this graph to a finite one. Nevertheless, it may still grow very huge, for instance due to multiple nested loops. To avoid this problem a mechanism has been integrated which prunes highly unlikely execution paths.

6.4.1 Probabilistic Unfolded Workflow Graph

A probabilistic unfolded workflow graph (PU-graph) $G_{PU} = (V, E, \delta)$ must be acyclic and is like the regular probabilistic timed graph (PT-graph) defined by means of probabilistic nodes and probabilistic edges. As it will become clear there are some structural differences between PT and PU-graphs; which problems they raise and how to resolve them is explained in Section 6.5.

Before implicit temporal information can be calculated, the PU-graph must be generated by unfolding the given PE-graph. For a better differentiation between PE and PU-graphs, the nodes of an unfolded graph are from now on called hit-versions. Hit-versions are generated from PE-nodes during the unfold-procedure, along with their hit-dependent duration and branching Information stored in the according tables:

- Every hit-version $n^h_v$ of a PU-graph refers to an according extended node $n$ of an PE-graph.

- For the type of a hit-version the following always holds: $n^h_v.t = n.t$.

- The duration of a hit-version $n^h_v$ stems from the corresponding entry in the duration table of the extended node $n$, such that: $n^h_v.d = n.DT.getDuration(h)$.

6.4.2 Unfold of Conditional Structures

Algorithm 6.1 generates an unfolded graph $PU$ according to a given extended graph $PE$, which is the first input-parameter. The second input-parameter is the critical mass: it controls unfolding in order to limit the size of the unfolded graph and avoids infinite structures. Unfolding will stop if $curMass \geq criticalMass$. $curMass$ depicts the sum of execution-probabilities of already unfolded hit-versions of the end-node; basically it describes the current degree of unfolding compared to a fully unfolded graph with an unfold-degree of 100%.

During unfold single hit-versions will be expanded, which means that the unfolded graph is extended with new hit-versions of successor-nodes and according edges — as
described in Algorithm 6.2. A hit-version which has not yet been expanded is still open and therefore stored in the set openNodes. Figure 6.5 shows an extended graph along

with branching tables for decision nodes. It has been unfolded to the graph in Figure 6.6 using a criticalMass of 95% (execution probabilities of nodes are displayed on top of the nodes). Note that or-joins have been omitted to save some space. As one can see the unfolding stopped after the generation of the third hit-version of the end-node, as the \( \text{curMass} = \text{end}1_x + \text{end}2_x + \text{end}3_x = 0.963 \) exceeds the critical mass.

**Algorithm 6.1 Graph Unfold**

**Input:** Probabilistic extended graph PE  
**Input:** The criticalMass  
**Output:** Probabilistic unfolded graph PU

1: \( PU := (\text{Nodes}_{PU}, \text{Edges}_{PU}); \text{Nodes}_{PU} := \emptyset; \text{Edges}_{PU} := \emptyset \)
2: \( \text{first} := n, n \in \text{Nodes}_E \land n.t = \text{start} \)
3: \( \text{first}1 := \text{createNewHitVersion}(PE, \text{nil}, \text{first}) \)
4: \( \text{first}1.x := 1.0 \)
5: \( \text{openNodes} := \{\text{first}1\}; \text{Nodes}_{PU} := \{\text{first}1\} \)
6: \( \text{curMass} := 0.0 \)
7: while \( \text{curMass} \leq \text{criticalMass} \land \text{openNodes} \neq \emptyset \) do
8: \( \text{curHitVersion} := \text{determineHitVersionToExpand} (\text{openNodes}) \)
9: if curHitVersion is not an endPar then
10: expand(curHitVersion, openNodes, PU, PE)
11: else
12: expandPar(curHitVersion, openNodes, PU, PE)
13: end if
14: end while

**Explanations to Algorithm 6.1 - Unfold.** Line 1: initializes the unfolded graph with empty sets of nodes and edges. 2: determines the start-node. 3: the method createNewHitVersion creates a new hit-version of the node first. The second parameter identifies the predecessor hit-version in the unfolded graph UP, which is nil as first must not have any predecessors. The method automatically determines hit and version according to the currently unfolded structure of the unfolded graph PU. In this case both are bound to be 1 as it is the first hit-version to be created. Additionally it sets the duration of the new hit-version according to the duration-table of the given node start using the determined hit1.
4: sets the execution-probability of this hit-version to 100%. 5: initializes the sets openNodes and NodesPU with the new hit-version. 6: initializes curMass with 0% as no end-node has been reached until now. 7-14: this while-loop represents the actual unfolding-part of the algorithm. The loop will exit if curMass exceeds the criticalMass or if no more open nodes are available. 8: determines the hit-version which shall be expanded; it is the element in openNodes with the highest execution-probability. 9: differentiates between regular (conditional) and unfold for parallel structures. 10: starts regular unfolding of UP by expanding curNode; openNodes will be updated as curHitVersion is no longer open, but its successors are (unless they refer to an end-node). For further explanations on expand see below. 12: controls unfolding in parallel structures.

Algorithm 6.2 Expansion of a node - expand(curHitVersion,openNodes,PU,PE)

Input: The curHitVersion to expand
Input: The set of currently openNodes
Input: The (partially) unfolded graph PU
Input: The probabilistic extended graph PE
Output: The expanded graph PU and the modified set openNodes

1: decompose curHitVersion to node^hit^ersion
2: for each succ ∈ node.Succ do
3:    branchProb := node.BT.getProbability(hit,succ)
4:    if branchProb > 0.0 then
5:        succHitVersion^Vh = createNewHitVersion(UP,PE,curHitVersion,succ)
6:        NodesPU := NodesPU U {succHitVersion^Vh}
7:        OpenNodes := OpenNodes U {succHitVersion^Vh}
8:        EdgesPU := EdgesPU U {curHitVersion=>succHitVersion^Vh}
9:        succHitVersion^Vh.x := curHitVersion.x * branchProb
10:   end if
11: end for
12: OpenNodes := OpenNodes \ {curHitVersion}

Explanations to Algorithm 6.2 - Expand. Line 1: decomposes the given hit-version to get access to its parts node, hit and version. 2: loops over all successors of the referring node. 3: determines the branching probability from the node to a successor for the given hit. 4: new successor hit-version will only be added if this branching probability is greater than zero. 5: creates the new hit-version of the successor node. The method also determines hit and version for the new successor hit-version according to the structure of UP. 6: adds it to the graph. 7: adds new hit-version to the list of open nodes. 8: creates a new edge and adds it to the graph. 9: calculates and sets the execution-probability of the new hit-version. 12: eliminates the current hit version from the list of open nodes.

6.4.3 Unfold of Parallel Structures

Algorithm 6.1 (line 9) differentiates between two different types of node-expansions: the regular expansion as explained above and the expansion of a node which is an end-par. As
already mentioned a parallel structure must be blocked (one entry, one exit), therefore it is called par-block. Every branch of a par-block, a par-branch, must itself be par-blocked. The workflow-graph in Figure 6.7 contains one par-block with two par-branches. A successor of an and-split, which is always the first node of a par-branch, is called start-par (nodes B and F), and a predecessor of an and-join, which is always the last node of a par-branch, called end-par (nodes E and I).

6.4.3.1 Problem Statement

Why is it so important to distinguish end-pars from other nodes? As already stated an and-split divides the workflow into multiple parts which are executed in parallel. The paired and-join synchronizes these parts as it demands that all par-branches (which implies all its end-pars) must be finished before it allows the workflow to continue execution. Therefore the time-interval and the execution-probability of an and-join will be determined by all its predecessors, contrasting or-joins where exactly one predecessor has to finish. Therefore
the unfold-algorithm must not generate an and-join until at least one end-par of each par-branch is available, in order to synchronize them.

### 6.4.3.2 Sketching the Idea

Figure 6.8 shows the expansion-order for the original graph in Figure 6.7. Note that or-joins have been omitted to save some space. Start-pars will always have the same execution-probability as their preceding and-split. At $E_{\#1}$ unfolding can not proceed with $AJ$ as the and-join has to synchronize it with the end-par $G$ of the second par-branch. Therefore it has to continue with the second par-branch in order to generate one. After $G_{\#1}$ the first synchronizing hit-version $A_{/\#1}$ can be generated. The execution-probability of an and-join is calculated as the product of execution-probabilities of its predecessors divided by the execution probability of the according and-split.\(^2\) Then unfolding continues with

---

\(^2\)This may not seem very intuitive at first glance, but is necessary for structures with multiple nested conditional and parallel structures.
ation of $E_{12}$. Now it generates $A_j$-hit-versions for every possible combination of $E_{12}$ and already generated end-pars of the other par-branch ($G_{11}$ and $G_{12}$), and continues unfolding with the one with the highest execution-probability which is $A_j$. Following this path to the end node $end_{13}$ results in the current mass 91.875%, which exceeds the critical mass for this example, therefore unfolding stops.

### 6.4.3.3 Algorithm

A prerequisite for Algorithm 6.3 is the global data-structure $ParInfo$ which stores information about already unfolded end-pars of certain par-blocks and branches. Assume that at each expansion of an and-join $aj_i h_o$ a tuple $parBlock = (aj_i h_o, ParBranches)$ will be added to $ParInfo$ – this has been omitted in Algorithm 6.3 to reduce the complexity of the pseudo-code. $ParBranches$ holds a set of elements $parBranch = (startPar_{ij}, EndPars)$ for each par-branch which is identified by its first node $startPar_{ij}$. The associated set $EndPars$, which is initially empty, will contain hit-versions of already unfolded end-pars of this branch. It is a necessity that each hit-version of an and-join must have its own $ParInfo$-structure as par-blocks may be nested or reside in cycles. The table in Figure 6.8 shows the $ParInfo$ ele-

### Algorithm 6.3 Expansion of an End-par – expandPar(curHitVersion, openNodes, PU, PE)

**Input:** The node/hitversion to expand $curHitVersion$  
**Input:** The set of currently openNodes  
**Input:** The (partially) unfolded graph PU  
**Input:** The probabilistic extended graph PE  
**Output:** The expanded graph PU and the modified set openNodes

```plaintext
1: decompose curHitVersion to node$^\#hitversion$
2: determine andSplit$^HV_{cur}$ and startPar$^HV_{cur}$ for curHitVersion
3: determine parBranch$^cur$, such that $(andSplit$HV$_{cur}$, parBranch$_{cur}$) $\in$ ParInfo
4: determine EndPars$^cur$, such that $(startPar$HV$_{cur}$, EndPars$^cur$) $\in$ parBranch$^cur$
5: OtherEndPars $= \{EndPars | (aj, EndPars) \in parBranch_{cur} \setminus endPars_{cur}\} $ if not(\$\{aj, EndPars\} \in OtherParBranches \land EndPars = \emptyset$) then
7: determine successor node andjoin, such that andjoin $\in$ node.Succ
8: AllCombinations := determineP pulverizationSets(OtherParBranches)
9: for each Combination $\in$ AllCombinations do
10: andjoin$^h_o$ := createNewHitVersion(PL, PE, curHitVersion, andjoin)
11: andjoin$^h_o$.x := 1
12: OpenNodes := OpenNodes $\cup \{andjoin$^h_o\}$
13: Combination := Combination $\cup \{curHitVersion\}$
14: for each endParHitVersion $\in$ Combination do
15: EdgesPu := EdgesPu $\cup \{endParHitVersion => andjoin$^h_o\}$
16: andjoin$^h_o$.x := andjoin$^h_o$.x $*$ endParHitVersion.x
17: end for
18: andjoin$^h_o$.x := andjoin$^h_o$.x / andSplit$^HV_{cur}$
19: end for
20: end if
21: OpenNodes := OpenNodes $\setminus \{curHitVersion\}$
22: EndPars$^cur$ := EndPars$^cur$ $\cup \{curHitVersion\}$
```
Explanations for Algorithm 6.3 - ExpandPar. Line 1: decomposes the given hit-version to get access to its parts node, hit and version. 2-4: determines the parBlock\textsubscript{cur} and parBranch\textsubscript{cur} stored in ParInfo to which curHitVersion belongs. The membership of a hit-version to a certain par-block or par-branch can be determined by searching the unfolded graph backwards for the next hit-versions of start-pars and and-splits. 5: creates a set of all par-branches which are stored in the parBlock\textsubscript{cur} except parBranch\textsubscript{cur}. 6: checks if every par-branch in OtherParBranches contains at least one end-par hit-version, as the and-join must synchronize end-pars from all par-branches. 6: determine the and-join, which is the single successor of the node to which curHitVersion refers to. 7: creates a set of all end-par combinations, such that each Combination contains exactly one endPar from each parBranch in OtherParBranches. 9-19: this nested loop creates a new and-join hit-version for every Combination of end-pars, connects them with new edges and sets the according execution-probabilities. Please also note that the new hit-version is added to the set of open nodes (line 12) and the curHitVersion is added to each combination (line 13). Finally the current hit-version is removed from the set of open nodes and added to the list of already visited end-pars in the relevant par-branch (lines 21 and 22).

6.5 Calculation of the Probabilistic Timed Graph

The time management calculation operations presented in previous chapters were originally designed for probabilistic timed graphs that structurally differ from an unfolded graph. Therefore the following differences must be considered when calculating or querying E and L-histograms.

- Or-joins are no longer synchronizers of multiple paths as they will always have exactly one incoming edge.
- Activities which are end-pars have multiple outgoing edges.
- The unfolded graph contains multiple versions for one specific hit of an and-join – one version for each end-par combination.
- The unfolded graph has multiple end-nodes.
- The unfolded graph may contain open nodes.

6.5.1 Forward Calculation

The forward-calculation operations defined in Chapter 4 must be applied to determine the E-histograms for each node, where the above mentioned structural peculiarities of the unfolded graph must be considered as follows:
- **Or-joins** An or-join will always appear in a simple sequence of nodes and must therefore be treated like a regular activity. Another option is to omit or-splits during the unfold procedure.

- **End-pars** These activities are to be treated like regular and-splits. The fact that different versions of succeeding and-joins with different execution probabilities exist has no influence on the calculation of E-histograms.

- **And-joins** And-joins in unfolded graphs must be treated like and-joins in regular timed graphs. The fact that different versions with different execution probabilities exist does not influence the max-conjunction operation. During run time it is necessary to synchronize the execution at the and-join, by selecting the correct version of the specific and-join hit – dependent on the paths that have been chosen in other par-branches; this can only be done when all par-branches have finished execution.

- **Branching Probabilities** It is quite obvious that the calculation of forward branching probabilities can be omitted as or-joins may only occur in sequences, and EPE-histograms of synchronizing and-joins are calculated by applying the regular maximum conjunction, which gets by without branching probabilities.

- **Multiple End Nodes** Their existence does not raise problems for the forward calculation. The overall process duration must be calculated by aggregating the EPE-histograms of all existing end-node hit-versions (see Section 6.5.3).

- **Open Nodes** Their existence does not raise problems for the forward calculation at build time. But they must be considered at run time – when process execution reaches a state which is represented by an open node the unfold procedure along with time calculation must be dynamically started from this node. Please refer also to explanations about dynamic adjustment in Section 3.6.3.4. As dynamic adjustment is time and resource consuming one should aim to avoid it – for example by increasing the critical mass.

### 6.5.2 Backward Calculation

The backward-calculation operations defined in Chapter 4 must be applied to determine the L-histograms for each node, where the above mentioned structural peculiarities of the unfolded graph must be considered as follows:

- **Or-joins** An or-join will always appear in a simple sequence of nodes and must therefore be treated like a regular activity. Another option is to omit or-splits during the unfold procedure at all.

- **And-joins and End-pars** End-pars merge paths from multiple succeeding versions of one specific and-join hit each with its own execution probability. Therefore end-par
LAE-histograms must be calculated by applying the weight and disjunction operations on LAS-histograms of succeeding and-join hit-versions. This approach corresponds to the backward calculation of or-splits in regular timed graphs, as it basically adheres to the same semantics. The necessary backward branching probabilities for the weight-operation are calculated as follows:

\[
p_{ep \rightarrow aj} = \frac{aj \cdot x}{ep \cdot x}
\]

assuming that an end-par ep in the unfolded graph precedes \( n \) hit-versions of an and-join \( aj \), \( 1 \leq i \leq n \).

- **Backward Branching Probabilities** Apart from the above-specified special case, there are no further adjustments necessary.

- **Multiple End-nodes** The calculation must be started by initializing the LAE of every end-node with the workflow deadline; in other respects the existence of multiple end-nodes poses no further problems to the calculation algorithm.

- **Open Nodes** Additionally an unfolded graph may have open nodes which are successors of or-splits or decision nodes (e.g. node \( F|I_2 \) in Figure 6.8). The problem is that the LAE for these open nodes cannot be initialized, as it is determined by the path originating from an end-node. One has to differentiate between two cases:

  1. The or-split has at least one non-open node. In this case perform weight and disjunction for all non-open successors. The resulting LAE-histogram will have a probability sum less than 1, is therefore invalid, which must be normalized as follows:

    **Definition 43 (Normalize Histogram):** A time histogram \((p, t) \in T \) with \( \Sigma p \neq 1 \) is normalized to \((p', t) \in T' \) with \( \Sigma p' = 1 \) as follows

    \[
    T' = \{ (p', t) \mid (p, t) \in T, \ p' = p \cdot \frac{1}{\Sigma p} \}
    \]

    Eventually occurring rounding errors can be corrected by adding the still existing difference to 1, to the probability of the tuple with the highest probability-value.

  2. Sometimes even all or-split successors may still be open. In this case every node on the branch between this or-split and the last preceding or-split must be removed. The only remainder of this path is its first node, which must be marked as open.
6.5.3 Histograms and Execution Probability

Before querying time histograms of an activity or a specific hit of an activity it is necessary to aggregate histograms of hit-versions. For instance the question “When will activity B presumably start?” can not be answered on behalf of one single hit-version, all hit-versions of B must be taken into consideration. The same applies for the execution probability of an activity – it must be summed up from multiple hit-versions.

- Execution Probability
  The overall execution probability for one specific hit can be calculated as the sum of the execution-probabilities of all its versions. For the $h^{th}$ hit of node $n$ it is
  \[ n_{h}.x = \sum_{v=1}^{V} N_{h,v}.x \]
  for all versions $1 \leq v \leq V$, where $n_{h}.x \leq 1$. Consider the following example for node C:
  \[ C_{1}.x = C_{1,1}.x + C_{1,2}.x + \ldots + C_{1,V}.x \]
  The same can analogously applied on multiple hits of one specific node, therefore the execution probability of node $n$ is determined from all hits $1 \leq h \leq H$ is
  \[ n.x = \sum_{h=1}^{H} n_{h}.x \]

Note that an overall execution probability may exceed 100%; these values are quite hard to interpret and can not be generalized. E.g. the execution-probability of node $M$, nested in a simple loop, is 350%. Depending on the scenario this can be interpreted quite differently:

1. The reentrance-probability of the loop is 100% for the first three iterations, and 50% for the last iteration. Therefore one can state that the node $M$ will be executed at least 3 times.

2. The second scenario starts with an or-split, where the branch with the loop which nests $M$ will be selected with a probability of 70%. The loop itself will always be executed exactly 5 times – each iteration with a 100% probability. Therefore one must state that the node $M$ will be executed five times, if it is executed at all. The probability of not executing it, which is 30%, is defined by the other or-split branch.

However, even though the summed up execution probability is hard to interpret, it can still be used for the normalization of time histograms.
6.6 Complexity Considerations

- Histograms For temporal statements about one activity, the histograms of its hit-versions must be aggregated. E.g. activity X has three hits X\(_1\), X\(_2\) and X\(_3\) with different EPS and different execution probabilities. To make a statement about the EPS of X one has to weight the single EPS with the corresponding execution probability and merge them by applying the disjunction operation. This can be applied for every time histogram T of a single hit or a single node as follows:

\[
\begin{align*}
n^h.T &= \bigvee_{v=1}^{V} n^h.T_v \ast n^h.x_v = n^h.T_1 \ast n^h.x_1 \lor \cdots \lor n^h.T_V \ast n^h.x_V \\
n.T &= \bigvee_{h=1}^{H} n^h.T \ast n^h.x = n^1.T \ast n^1.x \lor \cdots \lor n^H.T \ast n^H.x
\end{align*}
\]

The resulting histogram may be invalid as the sum of probabilities can be less or greater than 1. Therefore the probabilities must be normalized such that their sum is 1. As a matter of fact the forward calculation for or-splits enters the unfolded model through a back-door.

In order to have faster access to hit and node-based execution probabilities and histograms, it is proposed to store these values in an artificial hit n^0. For a single hit h of node which has several hit-versions: n^0 aggregates the time management information for the \(n\)th hit of \(n\).

6.6 Complexity Considerations

It is obvious that the feasibility of the unfold-algorithm is directly related to the problem of complexity explosion (described in [88]), which of course is dependent on the structures and branching probabilities of the workflow graph.

6.6.1 Complexity Explosion

It is very difficult to make general statements about the complexity, therefore it is only possible to make some general considerations. Let's start with an artificial negative example: an activity nested in a repeat-until structure. Assume that the backward transition has a constant branching probability of 99.99%. Even for a low critical mass the size of the unfolded graph will grow extremely huge. Fortunately we did not encounter such a branching-behavior in @enterprise customer-workflows. Usually the number of loop-iterations did not exceed 10. From experiences gained up to now we can state that the relation between the number of nodes in an original and unfolded graph ranges from about 1:5 to 1:1000. This, of course, heavily depends on the structure, the branching behavior, and the critical mass.
6.6.2 Complexity Reduction

The first alternative which avoids the expansion of too many nodes is to decrease the critical mass. Furthermore it is important to notice that unfolding and time management calculations are conducted at build-time; at run-time only single hit-versions of currently executed activities of a workflow-instance must be loaded. Additionally the graph should be split into several parts according to sync-points if possible (as described in Section 3.6.3.4). Partial graphs can be calculated and used in isolation, which reduces the complexity enormously. Other means of coping with the complexity, like encapsulating parts of the process in complex activities and unfold them on-demand (when execution enters the block), are subject of current and ongoing research.
Chapter 7

Simulation and Evaluation

In order to evaluate proactive techniques it was essential to test and simulate time-managed processes in a role-based workflow environment. Simulations were carried out within a recently finished workflow simulation framework implemented by Hannes Eichner [40]. Further details about this framework can be found in Section 9.2. This chapter provides first encouraging results for a few workflow scenarios.

7.1 Objectives

For deadline-constrained processes there are usually two main objectives to be pursued: reduce the number of deadline violations, and reduce the tardiness of processes [4]. The simulation aimed at an examination of the impact of probabilistic time management on these measures. For the scenarios presented below the conventional prioritization strategies FIFO, SIRO and EDF were compared with probabilistic prioritization policies MPDV and LPS. Additionally the server strategies ProbAbort, a probabilistic early escalation approach, were evaluated.

7.2 Basic Assumptions

Workload As described in Section 3.6.4.3 time management strategies should be applied to already 'balanced' systems, that are neither too underloaded nor too overloaded, in order to optimize the number of deadline violations, the mean tardiness, or even both measures. Therefore, in a first step, we concentrated on stable scenarios with corresponding resource-capacities, branching probabilities, activity durations, and arrival frequencies, that produced average workloads between 65% and 85%. Additionally the deadline was defined corresponding to the following conditions and parameters: for a FIFO-strategy with an average workload of 80% a minimum of 90% of all process instances must not violate the deadline. For further comments and considerations on this issue please also refer to Section 7.5.
**Simulation Parameters** For each run specific inter-arrival frequency, activity processing times (pure execution time without queuing time), and branching probabilities were applied. The execution time of an activity as well as the branching behavior were assumed to be stochastic and based on specified distributions.

**Resources** For simulation purposes the model had to be extended with information about resources, which allows for the definition of participants and group-hierarchies, as well as assigning them to certain activities.

**Worklists** For participants' worklists pull-oriented group-semantics were applied, which means: for activity instances ready for execution a work item is generated in the worklist of every potential performer. Potential performers are identified by resolving the specified group-hierarchies. If a performer selects an item for execution it is removed from all worklists. It is assumed that the performer selects the work item at the top of her worklist. Such an approach is very common for group-oriented administrative or production workflows.

**Log Extraction** As probabilistic strategies aim at exploiting knowledge of past executions it was necessary to extract information from a workflow log. Logs were created in a prior simulation run for each scenario, that used a FIFO strategy for worklists, which produced 10,000 process instances. A log stores information about branching decisions, the point in time an activity instance is ready for execution (arrival in the worklists of potential performers) and the point in time it is finished (when it leaves the worklist of the performer). Branching and duration tables (this time duration-values include queuing times), needed in the PE-graph, are easily extracted from these logs.

### 7.3 Scenario 1

#### 7.3.1 Process Model and Simulation Parameters

The first scenario, presented in Fig. 7.1, was based on a acyclic process with several nested control blocks:

- Two groups of participants were specified along with assignments for activity execution. Each participant in a group is only allowed to perform the assigned activities.

- Branching probabilities are displayed on the edges of the process and the base distributions for processing times of activities are displayed Figure 7.2. Note that although these processing times are represented like regular duration histograms they depict the pure processing times without queuing times.

- Duration histograms and the branching behavior for the PE-graph were extracted from the log generated in the prior simulation run.
The probabilistic timed graph with E and L-histograms was calculated by applying a deadline of $\delta = 150$. Histograms were compressed to 10 tuples.

A simulation set, which denotes the simulation with a specific client (or server) strategy, consisted of 10 runs with 1000 process instances with a given process interarrival-frequency.

Each simulation set was simulated with varying workloads of (about) 65%, 70%, 75%, 80%, 85%, and 90%, generated with process inter-arrival frequencies described by exponential distributions (with corresponding means between 10 and 15).

### 7.3.2 Simulation Results

We conducted one simulation set for each prioritization strategy. Process instances were never terminated, even if they already violated the given deadline. Test runs showed that for the probabilistic client strategy LPS a selection-probability of 70% proved to be best. The results for the average number of deadline violations is displayed in Fig. 7.3 and the results for the tardiness-percentages are displayed in Fig. 7.4.
Details for the workload of (about) 80% are displayed in Table 7.1. They show the following measures for different client-strategies: turnaround time \((\text{tat}_{\text{avg}}, \text{tat}_{\text{min}}, \text{tat}_{\text{max}})\), the average number of process instances that finished within the deadline \((w/in \ DL)\), the average number of process instances that violated the deadline \((\text{viol \ DL})\) and the average tardiness percentage \((\text{tat}_{\text{avg}})\).

<table>
<thead>
<tr>
<th>strategy</th>
<th>(\text{tat}_{\text{avg}})</th>
<th>(\text{tat}_{\text{min}})</th>
<th>(\text{tat}_{\text{max}})</th>
<th>w/in DL</th>
<th>(\text{viol \ DL})</th>
<th>tard_{avg} %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>94,0</td>
<td>38,2</td>
<td>267,4</td>
<td>947,8</td>
<td>52,2</td>
<td>31,75%</td>
</tr>
<tr>
<td>FIFO</td>
<td>93,8</td>
<td>37,6</td>
<td>205,1</td>
<td>954,6</td>
<td>45,4</td>
<td>24,34%</td>
</tr>
<tr>
<td>EDF</td>
<td>89,4</td>
<td>37,3</td>
<td>184,9</td>
<td>972,5</td>
<td>27,5</td>
<td>14,97%</td>
</tr>
<tr>
<td>MPDV</td>
<td>95,1</td>
<td>37,4</td>
<td>172,4</td>
<td>981,7</td>
<td>18,3</td>
<td>10,56%</td>
</tr>
<tr>
<td>LPS-70</td>
<td>93,9</td>
<td>36,8</td>
<td>167,6</td>
<td>979,6</td>
<td>20,4</td>
<td>5,34%</td>
</tr>
</tbody>
</table>

Table 7.1: Scenario 1 – Simulation Details (Workload 80%)

We additionally applied a ProbAbort-strategy (server-side) with a threshold of 10% – process instances with a less-than 10% probability finishing within the deadline were terminated (note that this is actually a very simple form of the traffic-light model). The results are displayed in Table 7.2 (ProbAbort-strategy). It has an additional column \(\text{aborted}\), which
lists the number of early escalations. The mean tardiness is not displayed as late processes were aborted before they could exceed the deadline.

<table>
<thead>
<tr>
<th>strategy</th>
<th>tat_{avg}</th>
<th>tat_{min}</th>
<th>tat_{max}</th>
<th>w/in DL</th>
<th>aborted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>90,2</td>
<td>37,1</td>
<td>148,7</td>
<td>987,6</td>
<td>12,4</td>
</tr>
<tr>
<td>FIFO</td>
<td>90,2</td>
<td>36,6</td>
<td>149,3</td>
<td>994,1</td>
<td>5,9</td>
</tr>
<tr>
<td>EDF</td>
<td>87,3</td>
<td>37,8</td>
<td>143,6</td>
<td>995,7</td>
<td>4,3</td>
</tr>
<tr>
<td>MPDV</td>
<td>93,5</td>
<td>37,7</td>
<td>149,6</td>
<td>999,0</td>
<td>1,0</td>
</tr>
<tr>
<td>LPS-70</td>
<td>92,6</td>
<td>38,0</td>
<td>149,8</td>
<td>997,3</td>
<td>2,7</td>
</tr>
</tbody>
</table>

Table 7.2: Scenario 1 – Simulation Results for ProbAbort-10 (Workload 80%)

7.4 Scenario 2

7.4.1 Process Model and Simulation Parameters

The scenario consisted of a cyclic process structure containing several nested control blocks, which are to be executed by four different user-groups as presented in Fig. 7.5. The execution time of an activity was assumed to be stochastic, based on the distribution of processing times (without queuing time) specified in Fig. 7.6 and the branching behavior
specified in Fig. 7.7. For unfolding a critical mass of 100% was chosen (see also Fig. 7.8) –

![Workflow Graph and Resources]

the unfolded graph consisted of 138 nodes, not all of them are visible in the figure. For the calculation of the PT-graph a deadline of 250 was applied. Histograms were compressed to 10 tuples. The process inter-arrival frequency for each simulation set was described by an exponential distribution with mean 10 (which generated an average workload of about 80% with the FIFO-strategy).

### 7.4.2 Simulation Results

Again two sets of simulation runs are compared: one with a NoAbort-strategy and the other one a ProbAbort-strategy with a threshold of 10% -- process instances with a less-than 10% probability finishing within the deadline are terminated. The results are displayed in Table 7.3 and Table 7.4. Test runs showed that in this case for LPS a selection-probability of 50% proved to be best.
7.5 Evaluation and Discussion of Results

7.5.1 Benefits

Client Strategies According to the results the following can be stated: client strategies that are not deadline-oriented (SIRO, FIFO) produce, as expected, the highest number of deadline violations along with a high tardiness percentage. Compared to this the non-probabilistic EDF-strategy performs much better, still exceeded by the probabilistic strate-
ProbAbort brings improvements in combination with any client strategy. The number of processes which hold the deadline is tremendously increased in all cases at the cost of few aborted instances. ProbAbort even seems to have a synergetic effect in combination with probabilistic client strategies, as the number of deadline violations is tremendously increased in all cases at the cost of few aborted instances. ProbAbort even seems to have a synergetic effect.
extremely low in these cases. The early termination of a few late process instances saves enough capacity to avoid deadline violations for most of the remaining instances.

7.5.2 Limitations

**Workload**  Simulation runs showed that the application of time-management brings no advantages in systems where the workload is too low (below 65%) or too high (above 85%); in the latter case even the opposite applies. A low workload means that worklists are empty most of the time, therefore prioritization and sorting brings no benefits. In systems with a high workload time-management enters a vicious circle: usually a lot of processes are late; it tries to speed up these instance, and has therefore to slow down other instances; these slow instances endanger their deadlines, therefore they must be accelerated; and so on. This behavior produces even more deadline violations than a simple FIFO-strategy.

**Probabilistic Early Abortion**  The probabilistic early-abortion strategy brings major benefits, but its design is still too simple: first, it may terminate process instances too early, which deprives them of their chance to catch up. Second, the termination of late process instances which are already near to completion is not optimal (waste of already invested resources). And third, escalation-costs are not considered at all – one might prefer to risk deadline violation of a process instance with high escalation-costs over an early (expensive) termination.

**Least Proportional Slack**  Although LPS-x has the potential of reducing the tardiness, the results of this strategy heavily depends on the selection-probability \( x \). Until now we do not know how to determine a good selection-probability – we simply tried out different values and chose the best.

7.5.3 Outlook

**Further Scenarios**  Although first results are encouraging, it is still a fact that until now only a few scenarios have been simulated and evaluated – to make clear and valid statements a lot more scenarios, based on real workflows, must be looked at.

**Increase Robustness**  One of the future main objectives will be to increase the robustness of probabilistic strategies, by sensitizing server and client strategies, such that selection parameters, like the one for the LPS-strategy, are determined dynamically according to the current system load and state of execution. Moreover we assume that it is possible to exploit knowledge about the structure of the workflow (possible paths after the current activity) to further increase the benefits provided by probabilistic time management.
Upgrade Probabilistic Early Abortion  Furthermore we plan to improve the ProbAbort-strategy by making a termination decision dependant on the deadline violation-probability along with the current progress. This relates expected process remaining time to the overall process duration – as it is suboptimal to terminate a process in its early stage where it still may catch up. Finally we plan to integrate cost-based early escalation [85] into the ProbAbort-strategy as it is suboptimal to terminate a process in a late state where it has already produced a lot of costs.
Chapter 8

Probabilistic Time Management for Advanced Structures

As many commercial systems allow further control flow structures, we had to extend the modelling possibilities of our graphs in order to increase the acceptance of probabilistic time management. Therefore we examined additional enterprise-structures as well as advanced control flow patterns provided by van der Aalst et al. [109]. We aimed at two basic objectives:

- Reduce the advanced structures to basic control flow structures if possible, such that they can be integrated into the unfolded graph.
- Retain the original structure of the process if possible.

This chapter explains how to integrate diverse advanced control flow elements into the probabilistic model. A related-work section has been omitted as all existing time management techniques, let alone probabilistic or stochastic attempts, are limited to basic control flow structures. Not even the need for supporting advanced structures has been recognized in most published papers.

8.1 Additional Structures in @enterprise

The techniques presented so far allow for time-managed @enterprise processes which consist of the following structures and WDL-statements: sequence, parallel execution, conditional execution (namely or-splits and joins, if-then, and choice), blocked cycles (namely while, repeat, and loop), and arbitrary forward or backward transitions which includes arbitrary cycles (implemented by means of the goto-statement). By supporting these structures many @enterprise customer workflows can already be time-managed. However, their workflow definition language WDL features even more structures [48]. In the following solution-proposals for each of the missing cases are provided, such that they can be integrated into the unfolded graph.
8.1.1 Sub-process

The @enterprise statement call starts a synchronous sub-process which blocks the main process until the sub-process is finished (see Figure 8.1). For the treatment of sub-processes one can choose among the following solution alternatives.

- **Treat as Complex Activity** The whole sub-process can be replaced by one complex activity which acts as a black box for the (eventually external) sub-process. The activity call must be augmented with the duration-histogram of the sub-process (estimated or calculated). For this solution no further adaptations are necessary. The disadvantage of this approach is that during execution of the sub-process predictions are restricted to the generalized histograms of the call-activity, as for activities nested in the sub-process no histograms exist.

- **Replace with Sub-process Structure** Another possibility is to replace each call-activity in the main process with the whole sub-process structure. This is suboptimal in terms of reuse and will result in huge unfolded graphs when complex sub-processes are integrated.

- **Integrate Partial Graph** The best solution is to prepare a reusable unfolded graph for the sub-process. The procedure is equal to the one of the the split-technique described in Section 3.6.3.4.

![Diagram of sub-process](image)

**Figure 8.1: Sub-process**

8.1.2 Asynchronous Batch Processing

The WDL-statement batch starts an (external) batch-process in a non-blocking fashion: this means that the main process continues execution and later, when the batch-process has been finished, it will be synchronized with the main process.
Figure 8.2 shows a simple main process that calls an external process (or batch). According to the control flow and the time information the execution of the main process will be blocked at the receiving activity, as the external process will not be finished when activity C is finished. Therefore the main process has to wait until the external process is finished. This means that the longest path between the calling and the receiving activity determines the duration, which is equal to a parallel execution between an and-split and an and-join. This scenario raises two problems for time management: a) no explicit link between calling and receiving activity is defined and b) the response time or duration of the external process must be known. The solution is based on so called lower bounds constraints. A lower bound constraint is defined between a source and a target activity which are not necessarily adjacent in the graph. It defines a minimum time that must pass after finishing the source activity, before the target activity is allowed to start (see [34]). Therefore the process designer has to augment each process that contains asynchronous calls with a lower bound constraint between calling and receiving activity. The lower bound is defined by the duration (or response time) of the external batch as displayed in Figure 8.3. Note that the response time may also be represented by means of a D-histogram. Time management calculations can be performed using the basic calculation operations for and-split and and-join, where the branching activity must be treated like an and-split and the receiving activity like an and-join. For further details on how to handle synchronous and asynchronous external processes see also [39]. There is one additional scenario to consider: in an unfolded graph multiple hit-versions of source and target-activities, due
8.1.3 Branch

The WDL-statement *branch* denotes an intersection in the main process that starts a parallel process which is executed independently from the main process. This means that even if the main process already finished, the branched process will be executed until it reaches its end-node (see Figure 8.4). Branches will mainly be used to perform non-critical maintenance tasks. For time management purposes it seems to be reasonable to ignore the branch and its succeeding activities, as they are not process-critical. Nevertheless, for some cases it might be necessary to support time-managed branches, e.g. the branch calculates results which are required immediately after the end of the process. Such a dependent branch is a parallel part of the process which splits at the *branch*-activity and must be synchronized at the end of the process alike a parallel structure.

In the timed graph each *branch*-activity must therefore be treated as and-split. Additionally an artificial *and-join* must be introduced as predecessor of the end-node (of the main process), where every branch-path of the main process will be synchronized (see Figure 8.5).
8.1.4 Par-for

A par-for (parallel for) may encapsulate any sub-structure, e.g. a sequence of tasks. During process execution this sub-structure will be spawned $n$ times and each spawn will be executed in parallel. The number $n$ can be set during build time or run time. See Figure 8.6 for a visualization. For time management purposes only the run time behavior must be adapted, quite similar to regular parallel execution. At the start of a par-for the time manager spawns the execution pointer $n$ times. Each of these pointers indicates the progress of one specific parallel spawn (it points to the currently active node). As the PT-graph contains only one structure (nested in the par-for block) two or more pointers may point to the same node at the same time. At the end of the par-for the pointers must be merged to a single one.

8.1.5 Or-par

An or-par is a structure that starts with an and-split and ends with an or-join. When the fastest path reaches the or-join all tokens on slower paths will be terminated. This structure is a hybrid, as the split-node adheres to and-semantics and the join-node to or-semantics. The problem with this structure is that the abortion-decision is deferred, as it is made at the end of the structure, at the join-node! Until execution reaches the join-node it is not clear which path will be finished and which ones aborted. For a time-managed treatment of or-par the following issues must be considered:

- **Build Time** The only thing we can tell for sure is that all paths but the fastest one will be aborted. Therefore the fastest path determines the time values of the structure. The solution is to apply the minimum-conjunction at the join during the forward calculation, and the maximum conjunction at the split during the backward calculation (inversion of the regular parallel structure).

- **Run Time** In a regular parallel structure all currently active parallel siblings must be considered and the worst case determines the probability of a deadline violation or the process remaining time. For activities in an or-par we propose to use the best case of all parallel siblings.
8.1.6 Ad-Hoc Workflows

Ad-Hoc workflows are workflows without a predefined control flow structure. Example: a workflow that consists of 4 activities; during run time users are allowed to select any activity to be executed next, which results in arbitrary forward and backward transitions between these 4 activities. Basically ad-hoc workflows eliminate the applicability of time management, as the calculation operations are based on a constant predefined control flow structure. For the treatment of ad-hoc workflow one can still fall back on the following solution alternatives:

- **Suboptimal but simple** The whole process is presented as one complex activity with an overall duration (which may be mined or estimated) and an overall deadline. The remaining execution time can be calculated as: \( \text{duration} - \text{now} \). The resulting histogram represents various probabilities for remaining times, which can then be used to query/calculate the probability for deadline violations.

- **Better but complex** Use workflow mining approaches to determine a control flow graph [109] and then apply the standard algorithms.

8.2 Advanced Control Flow Patterns

To increase the general acceptance of (probabilistic) time management we started examining additional advanced control flow structures, identified by van der Aalst et al. [109]. Again we aimed at retaining the original structure of the process or, alternatively, reducing advanced patterns to basic control flow elements. The following presents a short description of identified patterns and subsumes our findings. For deeper matters and more detailed descriptions please refer also to the master thesis of Mario Lassnig [70].

8.2.1 Multiple Instances

Within one process instance multiple instances of one activity can be created, e.g. activity \( A \) will be executed three times in a row (not in parallel). Usually the successor node is not started until the last of these instances is finished. Note that this behavior, although somehow similar to loops, is not reflected in the control flow graph. Van der Aalst et al. differentiate between the following cases for multiple activity instantiation.

**Design Time Knowledge** In this case the designer knows how often the activity will be executed. For time management purposes one must create a sequence with the specified number of activity duplicates. Alternatively the activity may be encapsulated in a (for-next) loop with a constant number of iterations.
No Design Time Knowledge  In this case the designer does not know how often the activity will be executed. If a log is available this poses no problem for probabilistic time management as it operates on empirical data anyway. It is easy to extract the likelihood for a certain number of instantiations and unfold the activity to a block-structured cycle containing hit-versions of the activity. In the case that no log-data is available estimates have to be applied.

Run Time Knowledge  In this case the number of instantiations will be determined during process execution, but it will be known before the first activity instance is created. If a log is available it can be solved by applying the same technique as for 'no design time knowledge'. Another alternative is to adjust the graph at run time as soon as the number of instantiations is known.

Without Synchronization  According to [109] a new process instance will be spawned for each of the multiple instances. Later each of these process instances is treated and executed in isolation. Therefore this pattern poses no problem for probabilistic time management techniques.

8.2.2 Cancellation

Cancellation means terminating an instance, either of an activity or a process, before an end-node has been reached. Cancellation of the last remaining activity instance implies that the process instance is cancelled as well. Note that process instances will starve if activity instances are withdrawn that are needed for the completion of certain join-nodes. Basically one could state that cancellation must not be considered in deadline-oriented processes at all, as their main-objective is to reach an end-node within the deadline. However, as there are applications which are not deadline-oriented – for instance forecasting upcoming activities – it should at least be possible to calculate E-histograms: introduce a cancellation probability $cp$ for each activity that is prone to cancellation. The graph must be extended by inserting an or-split right before the according activity. The successors of this or-split are the activity itself, connected with a branching probability $1 - cp$, and an artificial end-node, connected with a branching probability $cp$. The existence of multiple end nodes in the extended graph poses no problems for the unfolding and calculation algorithms. Even backward calculation of L-histograms can be applied, by treating artificial end nodes like open nodes as proposed in Section 6.5.

8.2.3 Deferred Choice

A deferred choice is modelled as a split where several alternative paths are offered, but only one can be chosen for execution. Although it sounds similar, there is a big semantic difference to a regular or-split: at run time all alternative paths (or their first activities)
will be offered to the participant(s) in charge until the decision for a specific path is made. This decision immediately terminates the other alternative paths. Therefore the decision is called 'deferred'. The deferred choice can be modelled by a regular or-split with the appropriate branching probabilities (mined from the log). The delay caused by the deferred decision can be represented by the or-split's duration.

### 8.2.4 Multi Choice

Multi-choice is a pattern that supports conditional parallel execution. In contrast to the regular or-split it allows the selection of one, several, or even all paths to be executed in parallel. The decision which paths will be executed and which not, depends on a decision evaluated at run-time – therefore it is not known at build time. Nevertheless it is possible to extract probabilities for possible path-combinations. Two alternative solutions can be proposed:

1. The multi-choice is transformed into a structure that consists of one or-split followed by several and-splits, one for each possible path or path-combination (see Fig. 8.7). The branching probabilities for the or-splits are equal to the probabilities of path-combinations.

2. As the first alternative is prone to complexity-explosion problems we proposed an approach that gets by without graph-transformations, a new backward-calculation operation that aggregates the information stemming from possible path-combinations into one LAE-histogram for the multi-choice node. For details please refer to [70]. As for all types of split-nodes, no special forward-calculation operations are necessary.

During run time the multi-choice has to be treated like any other parallel structure, where several activity instances are executed simultaneously. Note that this view onto a multi choice does not include deferred decisions. A delay between the activation of branches would result in scenarios where some paths are partially executed or even finished before
other paths are chosen and started. It is assumed that there is one atomic decision that tells the system which path-combination must be executed.

### 8.2.5 Synchronization and Multiple Merge

Van der Aalst et al. identified several variants of join-nodes, which are used to merge or synchronize multiple parallel conditional paths. However, all solutions described below are solely applicable to blocked structures in combination with a preceding multi-choice. Solutions for non-blocked structures are hard to come up with, as all paths stemming from arbitrary combinations of preceding and-splits, or-splits, and multi-choices must be synchronized and merged. This raises several problems which are subject to further research.

**Synchronizing Merge**  This is the direct counter-part of multi-choice. It synchronizes all paths currently executed in parallel, such that multiple execution of succeeding activities is ruled out. Two solutions are offered, analogous to the ones of the multi-choice:

1. It can be transformed into a structure consisting of several and-joins followed by one or-join (see Fig. 8.7).

2. It is possible to retain the graph-structure and apply a special aggregating forward-calculation operation for the EPS-histogram as proposed in [70].

As for all types of join-nodes, no special backward-calculation operations are necessary.

**Multiple Merge**  Similar to the synchronizing merge it is applied as a closing structure for conditional parallel execution. The difference is that it aims at token multiplication – this means that for every incoming token the outgoing branch is immediately activated, which results in multiple execution of the succeeding activity. However, van der Aalst et al. constrain multiple execution of the successor node, such that at maximum one successor instance will be active at a time – the other tokens are queued to be activated as soon as this instance is finished. This is equal to the behavior of multiple instantiation patterns described above. As solutions to all multiple instantiation patterns have already been provided it can be stated that this structure poses no further problems.

### 8.2.6 N-out-of-M Join - Discriminator

The discriminator waits for the first $N$ tokens from incoming $M$ paths. Therefore it synchronizes only tokens from the $N$ fastest paths and terminates remaining tokens on slower paths. This raises problems similar to the or-par structure in @enterprise. Therefore we propose a solution for a blocked combination with multi-choice, similar to the one for the synchronizing merge: a special aggregation operation for forward-calculations that considers only the $N$ fastest combinations of time-histogram entries. For further details
please refer to [70]. During run time it can be treated similarly to an or-par, only that the
$N^{th}$-fastest process instance (instead of the fastest) determines the results.

### 8.2.7 Interleaved Parallel Routing

The interleaved parallel routing pattern (IPR) specifies a set of activities that are to be
executed in any *sequential* order. This implies that for $N$ activities $N!$ permutations of
possible execution sequences can be generated. As an unfold approach is intolerable the
only proposable solution is to treat the whole structure as complex activity. The duration
of this complex activity is determined by calculating the EPE of one of the sequences.

### 8.2.8 Milestone

An activity can only be executed if a specified milestone has been reached and has not
expired yet. Consider the following example: a process contains a parallel structure with
three paths. Somewhere on this paths there are three activities A, B, and C. A milestone
has been specified that constrains the execution of activity B such that it can only start
when A has already been finished and must be finished before activity C starts. Eder et al.
[34] already showed how such temporal constraints can be specified and applied in a timed
graph by using upper bound, lower bound, and fixed-date constraints. An adjustment of
their interval-timed techniques to time histograms is subject of ongoing research.
Prototypical Implementation and @enterprise-integration

During the course of this thesis several prototypes had to be implemented for the evaluation of proposed techniques. Furthermore it was necessary to implement a simulation framework to verify the usefulness of the probabilistic approach and the advantages gained by it. And last but not least probabilistic time management features had to be integrated into the commercial workflow system @enterprise.

9.1 Prototypical Implementation

To test and evaluate histogram and probabilistic calculation operations several prototypical Java-based components were developed: an XML-based process model, an unfolding component, a calculation component and a viewer to check the resulting graphs and time histograms.

9.1.1 Process Model and Timed Graph Calculation

9.1.1.1 Probabilistic Extended Graph

The PE-graph is modelled by means of XML-structures and consists of two parts: a process model and exchangeable data-sets that describe the timing and branching behavior of the process.

**Process Model** The XML-structure is based on the definition of a workflow graph presented in Chapter 2.8. It consists of nodes and edges. A node is described by a name and a type, whereas an edge is described by a successor and a predecessor. The model supports all basic node types and allows all conformance classes described in Chapter 2.8.
Data Set A data-set contains hit-based information about durations histograms and decision tables to describe the branching behavior, as specified in Chapter 6. Nodes that are not listed are assumed to have a duration of 0 and edges that are not listed are assumed to have a branching probability of 100%. Data specified in this file can be easily extracted from a workflow log. Each process model may be linked to several data-sets.

9.1.1.2 Graph Unfolding

The result of unfolding the PE-graph is the PU-graph: an acyclic graph, as presented in Chapter 6, where each hit-version has exactly one D-histogram and one branching behavior, corresponding to its hit and version. For easier mapping between nodes and hit-versions the attribute enode has been introduced, which depicts the original node in the extended graph. The input parameters for the calculation of a PU-graph are:

- The PE-graph (probabilistic extended graph).
- The critical mass.
9.1 Prototypical Implementation

- A maximum number of hit-versions to unfold – this proved to be necessary for some (worst) cases as laid out in Section 6.6.

- The maximum number of tuples for histogram-compaction and a percentage for the histogram-cut operation. D-histograms will be compressed if they contain more than the allowed number of tuples.

- A limit for internal representation and rounding of decimal places.

The output of the unfold procedure is an intermediary XML-structure which contains additional information about the unfold result: number of current nodes, end-nodes and open nodes, as well as the current mass (unfolding degree). Probabilities are represented with the maximum number of decimal places (three dots indicate further decimal places).

```
<unfoldedgraph name="UnfoldedGraph-Workflow#1" created="03-10-2006 14:20:12">
  <basedon graph="Workflow#1" dataset="set-03-10-2006">
    <compression compact="10" cutpercentage="0.000000...">
      <round decimals="50">
        <mass critical="0.950000..." current="0.959860...">
          <numnodes max="10000" current="101" endnodes="12" opennodes="7">
            ...
          </numnodes>
        </mass>
      </round>
    </compression>
    <node name="D#1.1" hit="1" version="1" enode="D" type="ACTIVITY" />
    <node name="D#2.1" hit="2" version="1" enode="D" type="ACTIVITY" />
    <edge predecessor="D#1.1" successor="OS#1.1" />
    <edge predecessor="D#2.1" successor="OS#2.1" />
    <timehistogram refersToNode="D#1.1" type="DURATION" granularity="MINUTES">
      <histogramentry probability="0.500000..." time="12" />
      <histogramentry probability="0.500000..." time="15" />
    </timehistogram>
    <timehistogram refersToNode="D#2.1" type="DURATION" granularity="MINUTES">
      <histogramentry probability="0.250000..." time="7" />
      <histogramentry probability="0.750000..." time="5" />
    </timehistogram>
    <decisiontable refersToNode="OSl#1.1">
      <decision probability="0.700000..." successor="C" />
      <decision probability="0.300000..." successor="D" />
    </decisiontable>
    <decisiontable refersToNode="OSl#2.1">
      <decision probability="0.000000..." successor="C" />
      <decision probability="1.000000..." successor="D" />
    </decisiontable>
    ...
  </basedon>
</unfoldedgraph>
```

9.1.1.3 Probabilistic Timed Graph Calculation

The calculation component loads the XML-specification of the PU-graph and calculates the PT-graph based on a deadline (as described in Chapter 4 and Chapter 6). Furthermore
it provides calculation rules, histogram compression operations, granularity conversions, and probability rounding operations (to guarantee valid time histograms). The input parameters for the calculation of a PT-graph are:

- The PU-graph (probabilistic unfolded graph).
- A deadline.
- The maximum number of tuples for histogram-compaction and a percentage for the histogram-cut operation.
- A limit for internal representation and rounding of decimal places.

The output of the calculation procedure is again an XML-based representation which provides histograms and execution probabilities for every node and forward/backward-branching probabilities for all edges. In the example the calculation was performed on basis of the PU-graph presented above. Histograms were compacted to contain at most 10 tuples, whereas the compression operation cut was deactivated by specifying a percentage of 0%. The deadline of 200 is stored in the LAE-histogram of the last node and the overall process duration in the EPE-histogram of the last node. For rounding of probability values the number of decimal places to be considered and stored has been set to 50.

```xml
<timedgraph name="TimedGraph-Workflow#9" created="03-10-2006 14:23:09">    <basedon unfoldedgraph="Unfolded-Workflow#9" graph="Workflow#9"      dataset="set-03-10-2006">        <compression compact="10" cutpercentage="0.000000...">            <round decimals="50">                <createavg node="no" hit="no">                    <node name="BEGIN#1.1" hit="l" version="l" enode="BEGIN" type="START"                        xp="1.000000..." />                    ...                </createavg>                <edge predecessor="BEGIN#1.1" successor="A#1.1" fw="1.000000..."                        bw="1.000000..." />                ...            </round>        </compression>    </basedon>    <timehistogram refersToNode="BEGIN#1.1" type="DURATION" granularity="MINUTES">        <histogramentry probability="1.000000000000..." time="0" />    </timehistogram>    <timehistogram refersToNode="BEGIN#1.1" type="EPS" granularity="MINUTES">        <histogramentry probability="1.000000000000..." time="0" />    </timehistogram>    ...</timedgraph>
```

Two additional input parameters were added: createavg-node and createavg-hit which control the creation of additional 'virtual' nodes along with average execution probabilities and histograms containing average weighted values, as described in Section 6.5. E.g. with createavg node="true" the artificial node D90 will be added along with average execution probabilities and histograms of all existing hit-versions of D. Please note that artificial nodes are not connected by edges.
9.1.2 Process Viewer

The Process Viewer has been developed to verify and represent the results of process modelling, unfolding and probabilistic calculation operations. It contains the following features:

- Load and view extended graphs.
- Trigger unfolding and calculation of probabilistic timed graphs.
- View probabilistic timed and unfolded graphs and provide automatic layouting.
- View time histograms for nodes in the graphs.

The screen-shot in Fig. 9.1 shows a graph along with the EPS and LAS-histogram, and the one in in Fig. 9.2 shows a cyclic graph and its unfolded version.

![Graph and Start-Histograms](image)

**Figure 9.1: Process Viewer – Graph and Start-Histograms**

9.2 Process Simulator

To evaluate probabilistic time management and to test new ideas we developed a simulation framework. We chose to build on an existing open source Java-based simulation framework called DESMO-J\(^1\), which stands for Discrete Event Simulation Modelling in Java. It is an object-oriented framework for discrete event simulation and has been developed and implemented at the University of Hamburg. For further details about the implementation and applied simulation theory please refer to the master thesis [40].

\(^1\)[http://www.desmoj.de]
9.2.1 Extensions to the Process Model

For simulation purposes the above-mentioned implementation of the probabilistic model must be extended with information about resources. The extended model allows for the definition of participants and group-hierarchies, as well as assigning them to certain activities. They are specified by means of an additional XML-structure that contains information about participants, groups, and task assignments – as presented in this example:
9.2.2 The Simulation Framework

Worklists and Client-Strategies For worklists we aimed at pull-oriented group-semantics – if an activity instance is ready for execution a corresponding work item will be generated in the worklist of every potential performer. Potential performers are identified by resolving the specified group-hierarchies. If a performer selects an item for execution it will be removed from all worklists. Such a worklist behavior is very common in group-oriented administrative Workflow Systems. Therefore we chose to integrate configurable (generic) sort-policies to be used for a simulation scenario. Currently the system supports FIFO, SIRO and EDF, as well as the probabilistic policies MPDV and LPS.

Server Strategies Additionally we provided open interfaces for server-side time management strategies which allow the configuration of how to deal with process instances. Currently NoAbort (continue even late processes), Abort (terminate if deadline violated) and ProbabilisticAbort (terminate if deadline is foreseeable) are supported.

Simulation Parameters For each simulation run for a specific process scenario an initially specified inter-arrival frequency, activity processing times (pure execution time, without queuing time) and branching probabilities have to be applied. The mean inter-arrival frequency is defined by an exponential distribution with given mean. The execution time of an activity as well as the branching behavior are assumed to be stochastic based on specified discrete distributions – similar to the information in a data-set and also specified in an XML-file. Additionally the following parameters must be specified: client strategy, server strategy and the number of processes to simulate. The screen-shot in Fig. 9.3 shows the user interface of the Experiment Runner.

Log Extraction As probabilistic strategies aim at exploiting knowledge of past executions it was necessary to extract information from a workflow log. Therefore we integrated a feature that allows to configure and simulate prior process runs in order to generate the log. Note that these initial test runs are not time managed. This log stores, among other
things, branching decisions and the point in time an activity instance is ready for execution (arrival in the worklists of potential performers) and the point in time it is finished (when it leaves the worklist of the performer). From this the framework extracts the necessary data-sets, containing hit-based branching probabilities as well as duration histograms for activities.

**Unfold and Calculation** Before simulation starts, the probabilistic unfolded and timed graphs are calculated. During simulation each process instance is synchronized with a timed graph.

**Simulation and Reporting** The screen-shot in Fig. 9.4 shows the user interface during run time. The report (see also Fig. 9.5) generated after each simulation run shows details about relevant measures, which are for instance: average turnaround times, throughput, workload, the number of deadline violations and the tardiness of processes instances as well as details about selection behavior (queues) of participants.

### 9.3 Probabilistic Time Management in @enterprise

In 2005 we, Johann Eder and myself, started a research project in cooperation with Groiss-Informatics\(^2\) with the aim of integrating probabilistic time management in their workflow system @enterprise. The project is partially funded by the FFG – 'Österreichische Forschungsförderungsgesellschaft mbH\(^3\), which is the central institution for funding and promotion of technology innovations in the area of applied research in Austria. Most infor-

\(^2\)http://www.groiss.com
\(^3\)http://www.ffg.at
9.3 Probabilistic Time Management in @enterprise

Information in this section stems from the preliminary manual 'Time Management @enterprise - Draft' by courtesy of Groiss-Informatics [53].

9.3.1 Introduction

The project started with a prototypical implementation of necessary data structures and algorithms, which applied research results on full-blocked processes, in order to evaluate the approach. It also included a run time mapper, to synchronize running processes with the timed graph, as well as rudimentary extensions for the process design component in order to allow designers to view and enter time management-related data. Some additional prototypical components had to be implemented, as for example a log-extraction tool, which mines probabilistic data from @enterprise process logs. Later we added graph-unfolding techniques in order to deal with the goto-statement frequently used in @enterprise process definitions. The project is now in its second year and probabilistic time management features will be part of the next @enterprise major version to be released in December 2006.
9.3.2 Process Mining

The system offers a mining component that allows the extraction of empirical knowledge from the \@enterprise-log. Figure 9.6 shows the opening dialog. As one can see it is possible to limit the time span of extracted log entries and to store the mined values in a unique data-set. The idea of limiting the time span and using different data-sets was introduced by our partners at Groiss-Informatics, based on the assumption that seasonal fluctuations may distort forecasts based on a single average data-set.

9.3.3 Process Editor

Time management features have also been integrated into the proprietary process editor. It enables the designer to load a data-set into the currently edited process and to view duration histograms and branching probabilities as displayed in Fig. 9.7. The designer is also allowed to edit time histograms as well as branching probabilities and to store the adjusted data-sets.
9.3 Probabilistic Time Management in @enterprise

9.3.4 Unfold Graph and Timed Graph

Via the editor it is possible to enact unfolding and timed graph calculation and to view the resulting histograms for activities as well as for the whole process (see also Fig. 9.8). Unfolded and timed graphs are stored in the database. The data-structures used to store temporal information are similar to the XML-based structures of the prototypical implementation. In order to apply time management features during run time they had to realize a component for calendar mapping and another one to synchronize running process instances with the timed graph in the database. To reduce memory utilization they only load the time histograms needed by currently executing activity instances.

9.3.5 Predictive Features

Current predictive features comprise forecasting of process durations as well as a traffic-light model for identifying late processes. The screen-shot in Fig. 9.9 shows the forecast on the process start for a given deadline (to be entered in the field 'Fertig bis:'). Note that per default the regular deadline specified for this process will show up, but it can be adjusted by the participant which starts the process. The background colors in the time-bar (red, orange, and green) represent the current temporal state. The vertical line in the time-bar represents the predicted end of this process instance, which is still in state 'green'. The histogram-blocks represent the frequency of prior process-durations based on the applied data-set.
9.3.6 Worklist

The worklist presented in Fig. 9.10 shows each work item along with its activity deadline – determined by the LAE-histogram of the adhering activity in the timed graph. By clicking on the deadline the participant can view further time management data, e.g. traffic light model and predicted process duration. By using this feature each participant can answer customer-inquiries regarding the end of the process instance. The color-code of the represented deadlines adheres to the traffic-light model. Work items marked red are late – the process administrator or the participant are warned and may perform actions to speed up the process.

9.3.7 Upcoming Features

Groiss-Informatics plans to introduce further time management features in the near future. Among them is a personal scheduling component, which adds information about upcoming activities to the work-list of each participant. Furthermore a resource planner which shows the process administrator details about the resource-utilization in the near future, for instance if a single participant or a group of participants will be overloaded. A component that aids resource-planning by providing optimized schedules based on forecasts of resource-utilization is currently under research.
9.3 Probabilistic Time Management in an Enterprise

Figure 9.8: Time Histograms

Figure 9.9: Predictive Features
Figure 9.10: The Worklist
Chapter 10

Conclusions and Further Work

This chapter summarizes and discusses findings, briefly describes current work, and gives an outlook on future work.

10.1 Conclusions

The main objective of this thesis was to introduce probabilistic time management for processes in order to deal with uncertainty during process execution. The basic hypotheses stated that the application of probabilistic techniques at workflow build and run time

- reduces the number of deadline violations,
- reduces the tardiness (amount of lateness) of time-constrained workflows,
- allows forecasts about upcoming activity assignments, and
- allows the prediction of process completion times.

The thesis showed that these hypotheses hold (at least for certain types of process scenarios) based on the description of diverse predictive techniques and the simulation of pro-active features. It can be stated that the integration of probabilistic time management has great potential as it brings major benefits (at least for certain types of process scenarios). In particular solutions to all initially requested objectives could be provided as follows:

Modelling and Representation of Uncertain Information  Every time property can be represented by means of time histograms. They were introduced to capture uncertain temporal information, along with cumulation techniques for querying them, and compression techniques for reducing their size. For the representation of uncertain branching behavior in conditional structures branching probabilities were introduced, which reflect path-selection choices to be expected during process execution.
Calculation Rules for Basic Control Flow Elements A probabilistic representation of implicit time properties and time constraints requires calculation rules which incorporate the execution semantics of basic process structures, namely sequential, parallel, and conditional control flow elements. The forward and backward calculation-rules provided in this thesis are used to determine time histograms for implicit time-properties and constraints, which are stored in the probabilistic timed graph for further usage.

Provide Solutions for Cyclic Process Structures The commercial workflow system @enterprise as well as several other systems support the definition of arbitrary cycles. This posed a severe problem for time-management algorithms as they operate solely on acyclic workflows. To resolve this problem a new conformance class – called par-blocked – has been introduced which constrains the process graph to certain structural patterns. This allows the application of a specially designed unfold technique. As an unfolded graph is acyclic and consists only of basic control flow elements, basic calculation rules can be applied with only minor adjustments. In addition the unfolded graph provides hit-based view on activities, along with their durations and branching probabilities since these properties tend to vary from iteration to iteration.

Provide Solutions for Advanced Control Flow Elements Another objective was to examine advanced control flow patterns and @enterprise control flow elements and to provide solutions for an application in a probabilistic model (if possible at all). This has been achieved for most advanced structures, although for some only when they are combined in a blocked fashion.

Holistic Time Management Approach As most existing techniques are focused on rather special time management issues, it was a necessity to provide the basis for a holistic time management approach that includes diverse predictive and proactive build and run time features, as well as a description of how to integrate them into a workflow management system architecture. Therefore this thesis proposes an architecture that allows the integration into a workflow management system, which supports all kinds of time management features.

Table 10.1 compares support and representation of temporal properties and structural support of existing time management approaches with the one provided in this thesis. Jasper/Zukunft and Bussler are not considered, as the corresponding publications do not provide enough details for a comparison. For further details on the listed publications refer also to Appendix A.
### Table 10.1: Comparison of Time Management Approaches

<table>
<thead>
<tr>
<th>Authors</th>
<th>Aims at Build</th>
<th>Run</th>
<th>Blocking (acyclic)</th>
<th>Basic Structures Seq</th>
<th>Par</th>
<th>Con</th>
<th>Cycles Block</th>
<th>Arbit</th>
<th>Adv. Struct.</th>
<th>Temporal Information Explicit Dur</th>
<th>Start</th>
<th>End</th>
<th>Temporal Information Implicit</th>
<th>Prob</th>
<th>Prog</th>
<th>Dead</th>
<th>Det</th>
<th>Det</th>
<th>Det</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kao, Garcia</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det</td>
<td>dead</td>
<td>det</td>
<td>det</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jasper, Zukunft</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Haimowitz, et al.</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>int</td>
<td>adv</td>
<td>int</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panagos, Rabinovich</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det</td>
<td>dead</td>
<td>det</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eder, et al.</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det</td>
<td>dead</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eder, Panagos et al.</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det</td>
<td>adv</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olivera, Marjanovic</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>int</td>
<td>adv</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bussler</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zhao, Stohr</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det</td>
<td>-</td>
<td>-</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kafeza, Karlapalem</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>int</td>
<td>adv</td>
<td>int</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zhuge, et al.</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>int</td>
<td>dead</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bettini, et al.</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>int</td>
<td>adv</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combi, Pozzi</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>adv</td>
<td>det</td>
<td>prob</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Son, Kim, et al.</td>
<td>+</td>
<td>-</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det</td>
<td>dead</td>
<td>-</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v.d. Aalst, et al.</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>prob</td>
<td>-</td>
<td>-</td>
<td>prob</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Li, Fan</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det</td>
<td>adv</td>
<td>int</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baggio, et al.</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>det</td>
<td>dead</td>
<td>det</td>
<td>det</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lu, Sadiq, et al.</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>int</td>
<td>adv</td>
<td>int</td>
<td>prob</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eder, Pichler, et al.</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>prob</td>
<td>dead</td>
<td>prob</td>
<td>prob</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thesis</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td>-</td>
<td>-/+</td>
<td>prob</td>
<td>dead</td>
<td>prob</td>
<td>prob</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Blocking**: conformance class restrictions for acyclic structures. **Basic Structures**: + = supported, -/+ = partial (suboptimal) support, - = unsupported. **Cycles/Block**: + = roll-out-conditional, -/+ = roll-out-sequential, - = not supported/not described. **Cycles/Arbitrary**: + = full support, -/+ = partial support (par-blocked), - = not supported. **Advanced Structures**: - = not supported, -/+ = support for (some) blocked advanced structures. **Temporal Information/Explicit**: representation is det = deterministic/average, int = interval, or prob = probabilistic. **Temporal Information/Implicit**: representation is det = deterministic/average, int = interval, or prob = probabilistic.
10.2 Current Work and Research Issues

Client and Server Strategies  The probabilistic prioritization strategies presented in this thesis are beneficial in an already balanced system, but they are suboptimal when applied to overloaded systems. In order to increase the robustness we are currently examining how to integrate information about the current work load and the expected future path into these strategies. In addition we are working on an improved early escalation mechanism that for instance includes knowledge about the cost of an early abortion of process instances, so that late process instances with high escalation-costs are less likely to be terminated than those with low escalation costs. Currently we are testing several ideas and evaluate their feasibility by means of simulation.

Resource Planning and Optimization  Furthermore a resource planner which shows the process administrator details about the resource-utilization in the near future, for instance whether a single participant or a group of participants will be overloaded. An enterprise-component that aids resource-planning by providing optimized schedules based on forecasts of resource-utilization is currently under research. The model to represent capacities of resources and the means to detect overloaded periods is already finished, and the conceptual draft for the calculation of optimized schedules is near to completion.

Service-based Application  Service-based inter-organizational process automatization is the next step in the evolution of workflows. As with regular workflows a slow service can have a disastrous impact on the overall process response time, and even worse, result in the violation of time constraints. Therefore we are currently working on techniques and architectures that allow the integration of probabilistic time management into this field. Some results have already been published, like an architectural proposal [38] as well as calculation rules for processes with asynchronous service execution [39]. Currently we are working on proactive dynamic techniques for late processes, that allow rescheduling by dynamically exchanging slow services with faster ones.

10.3 Future Work and Research Issues

Eliminate Structural Constraints  The techniques proposed in this thesis can be applied on any full-blocked, loop-blocked or par-blocked process. Although this is sufficient for enterprise as well as many other commercial workflow systems, we want to further increase the acceptance of probabilistic time management. Therefore we are currently working on adjustments to existing forward and backward calculation-operations for application in sound cyclic non-blocked structures.

Constraint Satisfaction  Until now the proposed models only allow the representation of simple deadlines. The next step is to extend the model so that it supports upper-bound,
lower-bound, and fixed-date constraints, and to develop techniques to check the satisfiability of all constraints. The existence of these constraints influences valid execution intervals, which must be adjusted accordingly. Basic techniques have already been provided by Eder et al. [35] – we plan to extend them so that they support time histograms.
A Survey of Origins, Related Work and Systems

This chapter starts with a description of workflow time management origins, followed by a thorough overview and discussion of existing literature. Finally it examines time management capabilities and features of a few commercial and research workflow systems.

A.1 Workflow Time Management Origins

Several authors, like [12, 76, 88], identified the following research disciplines as the ones with the highest degree of influence on workflow time management: temporal constraint networks, project planning methods and scheduling (in soft real-time systems). Apart from these basic origins, which are thoroughly discussed below, some other time-related research areas can be identified. For instance temporal databases with an emphasis on storage and retrieval of temporal data and reacting on constraint violations at run time; or techniques based on Petri-nets, which, similar to process simulation approaches, emphasize analyzing process performance measures and calculating performance bounds; or techniques that utilize methods from queuing theory; and finally there are approaches which blend techniques from several origins. However, none of these techniques can be applied to workflow time management without major adaptations.

A.1.1 Temporal Constraint Networks

Many workflow time management publications are concerned with time constraint satisfaction issues, and many of these are based on temporal constraint networks, e.g. [72, 50, 44]. TCNs are a thoroughly researched type of constraint network. They are used to specify temporal constraints between events, for instance between start and end events of activities. TCNs can be represented as graphs, where nodes correspond to events and edges to constraints between two events. Temporal constraints are either of qualitative (e.g. A before B, C after B) or, as in most workflow-oriented approaches, of quantitative nature (e.g. A 10 hours before B, C 5 minutes after B). The types of inter-relationships
and constraints are rather diverse; for instance techniques based on the interval algebra of James Allen [1] offer 13 different temporal constraints: before, meets, during, overlaps, starts, finishes, and their inverse versions after, met-by, contains, started-by, finished-by, as well as equals. As temporal information is often incomplete and imprecise most approaches represent temporal constraints, activity durations, and scheduled start and end times of activities as [min,max]-intervals, e.g. [50, 6].

Finding a solution for a constraint satisfaction problem means assigning values to variables, adhering to specified domains (usually a range of integer values), under consideration of defined constraints. The assignment of values to all variables is called an instantiation. A constraint is satisfied if an instantiation does not violate it, and an instantiation is consistent if it satisfies all constraints. For the generation of valid assignments and to tighten the bounds various methods exist, for instance adaptations of algorithms to solve the shortest path problem like Floyd-Warshall or Bellman-Ford [50]. In TCNs variables represent the (scheduled) start and end time of activities. A consistent instantiation assigns a point in time to each of the variables, such that every specified time constraint is satisfied. It is important to notice that there is often more than one possible consistent instantiation. These instantiations may also be used at run time to schedule tasks in a timely fashion such that no constraints are violated.

Accordingly [72] states that to find some or even all arrangements, which would be necessary to determine valid execution intervals, is one of the big problems in temporal reasoning. Therefore it is actually not possible to determine exact bounds like E and L-values, but just one or some possible valid scenarios for them. Furthermore [76] states that TCNs support neither conditional nor cyclic structures, as they only allow checking instance types in isolation. This poses a big problem for run time applications, as it leaves only two suboptimal solutions: 1) to consider all possible tasks and constraints, which results in over-constrained networks and rather ineffective schedules, and 2) to switch between instance types as soon as a decision has been made, which reduces predictive capabilities.

### A.1.2 Scheduling (in Soft Real-time Systems)

Scheduling basically aims at finding an execution plan – a schedule with release dates – for a given set of steps and machines at process instantiation or during run time [10]. Precedence or temporal constraints may be specified, which negatively affect the schedulability of steps. A temporal constraint is for example an overall deadline that limits the process duration. Furthermore due dates on single steps, used for scheduling in soft real-time systems, are often applied. Soft real-time processes are processes where temporal deadlines must be met, but in contrast to hard-real time systems a deadline violation does usually not result in unacceptable consequences (e.g. plane crash, core meltdown). Due dates on single steps are basically latest allowed end times, correspondingly they are either predefined, or propagated from a global deadline to each sub-task, as for example described in
The techniques described there are comparable to the backward calculation of latest allowed end times. Additionally one might specify resource constraints to define capacity requirements of steps on limited resources, which adds an additional level of complexity to the scheduling problem [68]. The main scheduling objective is to find a feasible assignment of steps to machines, such that all constraints are satisfied. Feasibility is usually based on a quantifiable optimization criteria, like minimizing the number of late jobs, or minimizing the makespan. An inherent, yet very valuable feature is that scheduling considers all currently running jobs, which implies that it takes the actual work load into consideration - but that also implies that it can not be used for build time purposes.

The authors of [4] state that workflow scheduling and regular scheduling share the same ideas, and propose to map concepts as follows: a process instance can be viewed as a job, an activity as a step of a job, a participant as a machine, and the order of steps is determined by the control flow model. At a first glance project scheduling techniques [68] seem to be suited, but they lack support for conditional structures. According to [4] it is therefore necessary to merge concepts of flow-job problems, where all jobs must follow the same sequence of steps, and job-shop problems, where each job has its own route. They also claim that uncertainties, caused by varying execution durations and conditional execution, clearly place the problem in the stochastic domain, and that results regarding stochastic problems are minority in the scheduling literature, which basically means that no feasible solution exists until now.

Furthermore they mention problems caused by the workflow-inherent mix of preemtive and non-preemptive activities, and the m:n-mapping between activities (steps) and participants (machines). This is also discussed in [76], which states that scheduling-related approaches tend to simplify the problem, as in many workflows a pull-oriented approach is preferred over a push-oriented approach. Therefore a participant will rather prefer to choose which task in his work-list she is going to execute next, instead of sticking to a given schedule. This quickly results in major deviations between the actual state of the running process-instances and their calculated schedules. This also hampers the applicability in systems with long-running preemptive tasks, usually to be found in production and administrative workflows.

A.1.3 Project Planning Methods

Many workflow time management approaches are based on project planning methods (PPM) like the critical path method (CPM) or the program evaluation and review technique (PERT). The amount of literature on PPM-techniques is vast, and as basic ideas, concepts, and features have already been described in Chapter 3, no attempt to summarize them is made here. At a first glance it seems that they are well suited for workflow time management, as PPMs support several build and run time features. It is for example possible to calculate a timed graph, containing information like the earliest point in time an activity can start and the latest point in time it must end, such that the deadline will not be vio-
lated. They enable the designer to check the satisfiability of deadlines at build time and to detect critical activities and paths. Based on the timed graph they allow monitoring the timely execution of project tasks, and forecasting eventually arising deadline violations in the case of a delay.

Although workflow and project management share several characteristics, PPM techniques can not be applied to workflows in a straightforward manner, as several differences and shortcomings can be identified [12, 88, 76]. First of all there are basic conceptual differences: projects are often one-time endeavors of a non-repetitive nature, with frequently changing plans (structures) during the project. In contrast to this workflows, especially production workflows, are highly repetitive, typically based on a static structure, and therefore predictable by nature [76]. The repetitiveness of workflow processes also implies that often several instances of the same process are active in parallel, which is usually not the case with projects. This also implies that PPM-based methods do not consider the current work load generated by multiple running process instances. Last but not least, as already mentioned, PPMs are based on precedence graphs which do not support conditional execution or loops. Further differences, that are irrelevant for time management issues, like data and task-assignment issues, are discussed in greater detail in [12].

A.2 Related Work

According to their origins some of the techniques presented below are solely applicable during build time, others during run time only, again others in both phases, and some use static and others dynamic approaches. Corresponding to the diversity of applied models and techniques their objectives are rather diverse, which implies that a solid categorization and comparison is very difficult to achieve. Therefore the overview concentrates on the following: it describes and discusses existing approaches, identifies supported control flow structures, examines the representation of temporal information, and how implicit time constraints are determined. Additionally it focuses on their capability of dealing with temporal and structural uncertainty, how the temporal information has been modelled, and how it is used at build and run time. This survey also shows that probabilistic and stochastic concepts are insufficiently supported, and that the structural support is mainly restricted to sequential, parallel and, if at all, conditional execution. The overview is roughly sorted by publication dates. Alternative overviews of workflow time management literature, which mainly concentrate on build-time-oriented time-constraint issues, can be found in [33], [76] or [49].

Table A.1 compares support and representation of temporal properties and structures of existing time management approaches. Jasper/Zukunft and Bussler are not considered, as the corresponding publications do not provide enough details for a comparison.
<table>
<thead>
<tr>
<th>Authors</th>
<th>Aims at Build</th>
<th>Run</th>
<th>Blocking (acyclic)</th>
<th>Basic Structures Seq</th>
<th>Par</th>
<th>Con</th>
<th>Cycles Block</th>
<th>Arbit</th>
<th>Adv. Struct.</th>
<th>Temporal Information Explicit</th>
<th>Temporal Information Implicit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Dur</td>
<td>Start</td>
</tr>
<tr>
<td>1</td>
<td>Kao, Garcia</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td></td>
<td>-</td>
<td>det</td>
<td>dead</td>
</tr>
<tr>
<td>2</td>
<td>Jasper, Zukunft</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>-</td>
<td>int</td>
<td>adv</td>
</tr>
<tr>
<td>3</td>
<td>Haimowitz, et al</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td></td>
<td>-</td>
<td>int</td>
<td>adv</td>
</tr>
<tr>
<td>4</td>
<td>Panagos, Rabinovich</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td></td>
<td>-</td>
<td>det</td>
<td>dead</td>
</tr>
<tr>
<td>5</td>
<td>Eder, et al.</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>-</td>
<td>det</td>
<td>dead</td>
</tr>
<tr>
<td>6</td>
<td>Eder, Panagos et al</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>-</td>
<td>det</td>
<td>adv</td>
</tr>
<tr>
<td>7</td>
<td>Olivera, Marjanovic</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>-</td>
<td>int</td>
<td>adv</td>
</tr>
<tr>
<td>8</td>
<td>Bussler</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>det</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>Zhao, Stohr</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td></td>
<td>-</td>
<td>int</td>
<td>adv</td>
</tr>
<tr>
<td>10</td>
<td>Kaefeza, Karpalalem</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td></td>
<td>-</td>
<td>int</td>
<td>adv</td>
</tr>
<tr>
<td>11</td>
<td>Zhuge, et al.</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td></td>
<td>-</td>
<td>int</td>
<td>dead</td>
</tr>
<tr>
<td>12</td>
<td>Bettini, et al</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td></td>
<td>-</td>
<td>int</td>
<td>adv</td>
</tr>
<tr>
<td>13</td>
<td>Combi, Pozzi</td>
<td>+</td>
<td>+</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td></td>
<td>-</td>
<td>det</td>
<td>adv</td>
</tr>
<tr>
<td>14</td>
<td>Son, Kim, et al</td>
<td>+</td>
<td>-</td>
<td>full-blocked</td>
<td>+</td>
<td>-</td>
<td>+/-</td>
<td></td>
<td>-/+</td>
<td>det</td>
<td>dead</td>
</tr>
<tr>
<td>15</td>
<td>v.d. Aalst, et al</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td></td>
<td>prob</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>Li, Fan</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>-</td>
<td>+/-</td>
<td></td>
<td>-</td>
<td>det</td>
<td>adv</td>
</tr>
<tr>
<td>17</td>
<td>Baggio, et al</td>
<td>-</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td></td>
<td>-</td>
<td>det</td>
<td>dead</td>
</tr>
<tr>
<td>18</td>
<td>Lu, Sadiq, et al</td>
<td>+</td>
<td>-</td>
<td>non-blocked</td>
<td>+</td>
<td>+</td>
<td>-/+</td>
<td></td>
<td>-</td>
<td>int</td>
<td>adv</td>
</tr>
<tr>
<td>19</td>
<td>Eder, Pichler, et al</td>
<td>+</td>
<td>+</td>
<td>full-blocked</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
<td>-</td>
<td>prob</td>
<td>dead</td>
</tr>
</tbody>
</table>

Table A.1: Overview and Comparison of Existing Time Management Approaches

Blocking: conformance class restrictions for acyclic structures. Basic Structures: + = supported, -/+ = partial (suboptimal) support, - = unsupported. Cycles/Block: + = roll-out-conditional, -/+ = roll-out-sequential, - = not supported/not described. Cycles/Arbitrary: + = full support, -/+ = partial support (par-blocked), - = not supported. Advanced Structures: - = not supported, -/+ = support for (some) blocked advanced structures. Temporal Information/Explicit: representation is det = deterministic/average, int = interval, or prob = probabilistic. Time constraints are dead = simple deadline, or adv = advanced time constraints. Temporal Information/Implicit: representation is det = deterministic/average, int = interval, or prob = probabilistic.
A.2.1 Ben Kao and Hector Garcia-Molina

Although [62] and [61] are not specifically workflow-related, they are often cited in workflow time management literature, as they introduced several beneficial time management concepts. The authors examine sub-task deadline assignment in the context of scheduling in distributed soft real-time systems. In particular they concentrate on solving serial and parallel sub task problems. These techniques aim at assigning deadlines (= latest allowed end times) for sequential and parallel sub-tasks in a real-time process, based on an overall process deadline.

They propose several strategies to determine these sub-task deadlines. The simplest one, to be applied if (expected) task durations are not available, is to assign each sub-task the same deadline as its super-task. If task durations are available they are calculated like latest allowed end times, which are implicitly determined by the process deadline and durations of succeeding tasks. To avoid that the current sub-task consumes the slack of the whole process they propose dividing the available slack among the remaining sub-tasks, either equally or in relation to their execution durations (see also Section 3.5.3.4). A sub-task deadline is assigned at run time, just before it is submitted for execution. Tasks are scheduled according to an earliest deadline first policy: sub-tasks with firmer deadlines are scheduled earlier. Correspondingly the privileged execution of these sub-tasks proactively avoids violations of the overall process deadline.

Their techniques are basically applicable in any process-oriented system during run time. However, similar to regular scheduling approaches, the applicability is reduced as it is based on the assumption that (expected) task durations are deterministic. The model allows for (full-blocked) sequential or parallel, but no conditional structures, which also excludes cyclic structures.

A.2.2 Heinrich Jasper and Olaf Zukunft

In [58] the authors explain how to implement advanced temporal features for a workflow system based on the event condition action (ECA) model of active real-time database systems. They briefly describe the basic ideas of predictive time management, like the definition of deadlines and the calculation of execution intervals corresponding to precedence constraints and duration estimations. But they neither explain how to calculate these intervals nor how predictive scheduling is accomplished. They only mention that this has to be implemented in an external conventional scheduler which uses a least-slack algorithm. This implies that their approach suffers from the same shortcomings as every other regular scheduling-based approach, as stated in Section A.1.2. Furthermore, they discuss general scheduling aspects and basic time-failures. However, this paper contains two very important contributions for workflow time management: an ontology of time and the proposal of using a linear, discrete, one side bounded, relative time model.
A.2.3 Ira J. Haimowitz et al.

The authors of [50] address the lack of temporal features in workflow systems and propose an approach to model, track and reason about time and temporal resources within health care workflows. They apply a graph-based temporal constraint network paradigm to represent and specify temporal constraints between start and end events of activities. Start times, end times, and durations of activities, as well as temporal constraints are represented as \([\text{min}, \text{max}]\)-intervals. They apply a version of James Allen’s [1] interval calculus, which allows the definition of diverse types of temporal constraints on and between events. For temporal reasoning over valid execution intervals of activities and to tighten the bounds they applied a shortest path algorithm. The bounds determined by this algorithms can be used as input for a scheduler, such that the workflow execution does not violate any temporal constraints.

Their approach aims at providing schedules that satisfy all constraints as well as to monitor the timely execution of instantiated processes. Like many TCN-based techniques they show how to model and deal with sequential and parallel execution, as well as how to specify task-hierarchies (nesting), but they do not consider conditional or cyclic structures.

A.2.4 Euthimios Panagos and Michael Rabinovich

The authors introduced the concept of predictive workflow management which builds on work in the real-time system domain. In particular they present two interrelated techniques: dynamic deadline adjustment [83] and early escalation [84] which they combine in [85]. The main idea is to predict upcoming deadline violations as early as possible in order to force early escalation which saves further consumption of resources for processes that are ‘hopelessly’ late anyway (see Sections 3.6.3.4 and 3.6.4.2). During run time they adjust activity deadlines (= latest allowed end times) to the current system load and force an escalation if the process is expected to be late. The higher the cost of an early process termination, the more confident one must be that a deadline violation will actually occur.

The authors define a full-blocked model, which allows sequential, parallel and conditional execution. For each activity they assign an average duration (which includes queuing time, average queue length), average escalation cost, and a deadline. Activity deadlines are not calculated, they must be specified by a business analyst. If only one global process deadline is available, they propose the calculation of activity deadlines according to [62, 61]. This technique is solely applicable at run time and does not consider conditional structures – in fact the early escalation algorithm ignores them, as it simply stops the forward examination at or-splits.

In [84] they consider an additional case: if at build time no global process deadline is specified, they propose to calculate it based on the critical path (similar to the calculation of EPE-values). This technique treats conditional and parallel structures alike – the (critical) path with the longest duration determines the deadline. Additionally they state
that the usage of statistics about the branching behavior could significantly improve this method. Loops are mentioned and it is proposed to roll them out to a sequential structure, assuming an average number of iterations.

A.2.5 Johann Eder et al.

In [88] the authors present ePERT: an extension to the net diagram technique PERT to compute valid execution intervals for each node in the workflow graph, based on the process structure, average activity durations and an overall process deadline – they calculate earliest possible and latest allowed values for each node. The paper also gives an overview over how to use this time information at build time (check deadlines) and during run time (reactive and proactive features). Finally they demonstrate how time management has been integrated in the workflow research prototype Panta Rhei [27].

ePERT extends PERT, which supports only sequences and parallel execution, such that it supports conditional structures. At build time it is unknown which conditional branch will be executed after an or-split, therefore they propose a method which considers only durations of the best (shortest branch) and the worst (longest branch) case. This means that for each implicit time constraint a best and a worst case must be calculated and stored, e.g., best and worst earliest possible end time of a node, which basically defines an interval between a minimum and a maximum time value. Although average activity durations are applied in the course of the paper, they propose the use of the best, worst, and median execution time estimates for activities. The $\beta$-distribution can be used to compute activity execution times along with the according variance. However, the authors do not show how to integrate this probabilistic feature. For loops they propose rolling them out to a sequence, according to an average number of iterations.

A.2.6 Johann Eder and Euthimios Panagos et al.

[35] builds on prior work by Eder et al. and integrates additional explicit time constraints into an ePERT-based model: upper bound, lower bound, and fixed-date constraints which allows to specify time constraints based on absolute calendar dates. They describe how to adjust the valid execution intervals of activities at build time, such that the fulfillment of constraints can be guaranteed. Additionally they propose the usage of a traffic light model at run time (see Section 3.6.3.2). Alike ePERT, their model allows for sequential, parallel and conditional execution; again they propose rolling out cyclic structures or treating them as single complex activities with an average duration.

In [34] they further extend the workflow model with optional activities and alternative structures. Optional activities may be skipped during run time, if the process is late. Unlike conditionals, all branches succeeding an alternative split are ‘equal’ in the sense that any of them may be executed, but only one at a time. If the process is late the system will
automatically prefer a shorter alternative branch over a longer one. Additionally they extended the temporal model: they store four thresholds, instead of only two (minimum and maximum), for each implicit time constraint, which improves the prediction-capabilities of the system during run time.

In [29] they show how to check time constraints specified between disparate conditional execution paths, by (partially) unfolding the workflow graph, which has been considerably extended in [49]. Finally [33] integrates explicit time constraints into a refined temporal model.

A.2.7 Olivera Marjanovic and Maria E. Orlowska

[75] lists the features an intelligent, time-aware workflow system should have: modelling and verification of temporal constraints, duration estimations, time monitoring, time simulation and personal time management (similar to personal scheduling described in Section 3.6.3). For their agent-based approach they additionally postulate cooperation and learning as necessary features. They outline the architecture of a temporal manager and roughly describe its functionality.

The techniques presented in [77, 76, 78] mix ideas and concepts from TCNs and PPMs. Very similar to the work of Eder and Rabinovich they aim at checking the satisfiability of time constraints at build time, as well as mapping and monitoring these constraints at run time; with some predictive and proactive features mentioned as possible applications. They provide their own version of a timed graph, which merges workflow-specific generalizations of the shortest path partitioning algorithm and the critical path method (CPM). Their model allows for relative and absolute deadlines, as well as relative time constraints. Activity durations are represented as either (precise) average values or [min,max]-intervals. They examine single instance types as well as a generalized view on all possible instances that contains implicit time constraints represented as [worst-case,best-case]-intervals. They allow the same control flow constructs as Eder et al., namely sequential, conditional and parallel structures, but do not restrict them to being full-blocked. Cyclic structure are not treated – it is proposed to encapsulate them into complex activities. They additionally provide graphical representations for durations and implicit time constraints, called duration and instantiation spaces, which are utilized to represent temporal properties and status during build or run time.

In [74] the concepts are refined by introducing an additional operational level on top of the basic control flow model. In the operational model instance types for individual tasks are distinguished, where each instance type has its own duration. The use of instance types makes sense as the duration of a task may vary considerably depending on the specific case, e.g. the time spent handling a claim depends on the importance of the customer. This allows more precise statements about the satisfiability of constraints and the current
temporal status. The structure of the operational model is not equal to the original flow model, as alternative structures must be introduced for different instance types, but it is still possible to map the two models.

In [92], featuring Shazia W. Sadiq as first author, temporal concepts are also mentioned, but the paper mainly deals with changes in the workflow specification over time (cf. workflow evolution).

A.2.8 Christoph Bussler

In [12] the author proposes the integration of workflow systems with project management tools to provide the functionality necessary for time management. He lists differences in concept, as already discussed in Section A.1.3. One of the major problems is that project management techniques are based on simple precedence graphs, which do not allow the modelling of conditional control structures or loops. Therefore Bussler proposes rescheduling whenever a decision has been made, like entering another loop or selecting a specific conditional branch. This implies resource-consuming repeated recalculations of the timed graph, as well as the impossibility of reasoning about implicit time constraints in cyclic structures at build or run time.

A.2.9 Leon J. Zhao and Edward A. Stohr

In [120] the authors describe a scheduling-based temporal workflow management application for claim handling systems. The objective is to optimize the turnaround time of process instances (claims). To calculate turnaround times they decompose the graph into clusters of conditional branches (between or-split and or-join). The turnaround of each cluster is calculated by weighting the durations of conditional sub-branches with a specified branching probability. During run time it is possible to determine the expected remaining time of the current process instance based on these clusters (similar to expected remaining times). They introduce several task prioritization policies. To enforce the use of a certain policy they propose different reward functions for workflow participants. For details please refer to Section 3.6.4.3.

For durations they differentiate between the (average) execution and queuing time of a task. Especially queuing times vary considerably, depending on the system’s work load. Therefore they extended their model with average queueing lengths, assuming that the state of the system is steady, which are used to calculate overall task durations. Additionally they propose a load balanced time allocation strategy, to be applied at run time, which calculates durations according to current queue lengths (see Section 3.6.3.4).

However their technique is run time-oriented. It works with a full-blocked workflow graph, which allows sequential and conditional, but neither parallel nor cyclic structures. The paper does not address explicit time constraints, which also excludes simple process
deadlines. They were among the first ones to deal with uncertainty during process execution: the introduction of branching probabilities allows a more differentiated view on possible future decisions. However, turnaround and remaining times are still calculated and represented as average values.

A.2.10 Eleanna Kafeza and Kamalakar Karlapalem

[60] shows how to specify temporal constraints on workflows, based on the interval algebra of James Allen [1]. Constraints are defined as temporal relations between activities. Temporal properties of activities are an average duration and a deadline. To calculate valid execution intervals, which guarantee process execution without constraint violations, they describe it as a constraint satisfaction problem. A process definition is constraint-consistent if there exists at least one execution that satisfies all constraints. The algorithm works with block-structured process models, allowing for sequences, and-blocks, and or-blocks, but no cyclic structures. Or-blocks are treated like and-blocks, as the worst case (longest path) is assumed. During run time a scheduler distributes tasks to agents, such that all temporal constraints are satisfied. They propose several types of schedulers. The first one operates solely on build time calculated model. The second one recomputes the model during run time in order to avoid rash and wrong escalation decisions, caused by uncertainties, like the varying duration or the selection of a shorter-than-expected branch at or-joins.

A.2.11 Hai Zhuge, To-yat Cheung and Hung-Keng Pung

In [121, 122] the authors propose an approach for workflow scenarios that are distributed over different time zones, e.g., in distributed team development applications. They describe how to calculate a time plan – activities scheduled in their valid execution intervals arranged over several time axis – such that the duration is minimized and all given time constraints are satisfied. To track the timeliness of the workflow during run time they define checkpoints at certain critical points, determine duration bounds between these check points, and show how to monitor these constraints. They propose rescheduling late processes which violate valid execution intervals but do not go into detail about this.

Their model basically consists of activity durations, flow durations (transition delays), and duration constraints between events, all represented by [min,max]-intervals. The (constant) temporal difference between each pair of time-axis must be known. Additionally the designer has to specify which activity will be executed on which time-axis. Activities are usually scheduled such that they must be executed during the working hours of a certain time axis. Their calculation rules focus on the determination of durations, always differentiating between possible placement-scenarios of activities on time-axis. The applied model allows for sequential, parallel, and conditional execution. However, they treat conditional and parallel structures alike (critical path approach) and they do not consider cyclic structures.
A.2.12 Claudio Bettini, X. Sean Wang and Shushil Jajodia

In [6] the authors propose enhancing the capabilities of workflow systems by specifying a temporal constraint network. As always with TCN-based techniques, a model is consistent if the constraint solver finds one valid assignment of execution intervals. At process instantiation they propose the calculation of free schedules, based on the build time calculated model, which aims at an execution without constraint-violations, by (pre)dispatching tasks to agents. It is assumed that each agent needs a specific amount of time, specified as [min,max]-interval, to execute a specific task, which is known in advance. This schedule also provides the means for predicting constraint violations during run time. Their model supports quantitative temporal constraints represented by [min,max]-intervals. It allows for relative time constraints between certain events, as well as for process deadlines and maximum activity durations. The model basically supports sequential, parallel and conditional, structures, but for conditional structures they examine each possible instance type in isolation — where an instance type consists only of sequential and parallel structures. Unfortunately they do not specify how to merge all instance models into one combined model, which is problematic for decisions at run time, as it is impossible to know in advance which instance type should be consulted after the next or-split. For loops they propose encapsulating them in complex activities.

The work in [7] is basically a more detailed version of the former paper, which additionally shows how to handle different time granularities.

A.2.13 Carlo Combi and Giuseppe Pozzi

In [19, 21] they demonstrate how basic temporal (mainly reactive) time management features are implemented for a relational database (Oracle). The functionality of this temporal layer resembles ECA-concepts in temporal active databases (cf. [58]).

In [20] they present a technique for temporal conceptual modelling of workflows, to be applied in soft real-time workflows. They introduce a non-blocked workflow model that allows the specification of sequential, conditional, and parallel structures, but no loops. They augment their model with basic temporal information, like activity durations, and different types of constraints, similar to the ones applied by Eder et al. or Marjanovic and Orlowska. They introduce a new type of time constraint, to be specified on and between groups of tasks, and differentiate between the expected duration and the maximum duration of a task. They allow different time granularities and propose transforming all temporal information to the finest granularity level. They describe calculation rules which narrow the temporal intervals for (expected) start and end events of tasks, based on all initially specified time and duration constraints. This procedure is actually similar to the calculation of a timed graph with E and L-values. The narrowed intervals define the input for a soft real-time scheduler. As they propose the calculation of schedules at
process instantiation their technique is solely applicable at run time. A major drawback is constituted by the fact that they treat conditional and parallel structures alike. They propose scheduling all tasks and simply skipping those tasks that are not executed due to conditional execution. The problem is that already scheduled temporal (and eventually resource) constraints may become void, unnecessarily constraining the schedule. Cyclic structures are not considered.

A.2.14 Jin Hyun Son and Myoung Ho Kim et al.

In [18, 98] the authors describe how to identify the critical path – the path with the longest duration – at build time. Their technique has its foundations in queuing theory. Each activity is described by an independent M/M/1 (single-server) queuing system (see also [10]). Service request arrivals form a Poisson process and service times are described by means of an exponential distribution. From this they determine the expected duration of activities and structural blocks. Activities are combined to a workflow queuing network, which is actually equal to a full-blocked workflow graph. It allows sequential, parallel, conditional, blocked cyclic structures, as well as variants of the advanced control structures multi-choice along with according join-elements. The longest path is determined from the inside out, starting with the duration calculation of the innermost control blocks. For the duration of conditional and parallel blocks the nested path with the longest duration is considered. Cyclic structures are rolled out to a sequences, according to a given distribution. In [96, 97] they utilize this model to introduce a static slack distribution strategy. They proportionally distribute the overall process slack – calculated as difference between the process deadline and the duration of the longest path – among activities and blocks residing on the longest path.

A.2.15 Wil van der Aalst et al.

In [111] the authors introduce discrete time stochastic Petri nets (dts-net), where the service time of a transition is represented as arbitrary discrete random variable, called service density – as a matter of fact this representation is similar to the duration histogram used in this thesis. The structure of the dts-net coincides with process-algebraic expressions, which allow conditional, parallel, sequential, and cyclic structures, but is restricted to blocked structures. The branching behavior of iterative and conditional blocks is described by a Bernoulli-distributed random variable. The presented technique aims, among other things, at the calculation of the throughput density, based on the service density of single blocks. The authors also state that it is possible to compute an arrival function for each single place in the net, similar to the E-histograms described in this thesis. As they do not consider explicit deadlines it is not possible to determine implicit time constraints, like the latest allowed end time.
Additionally the authors provide a thorough overview over different analysis techniques for all kinds of timed Petri nets. In these nets timing may be deterministic (average values), non-deterministic (interval representation), or stochastic. For example in general stochastic Petri nets (GSPN), which correspond to semi-Markov processes, that can also be utilized to calculate various performance measures.

[110] deals with the application of different proactive deadline-avoidance techniques. Details about the calculation of temporal information, like predicted completion times, is outside the scope of the paper, as well as how they apply it on the process model represented by a colored Petri net. They assume that the model contains no loops – they briefly recommend applying a worst case approach, considering only the last iteration. To detect possible upcoming deadline violations they propose comparing task deadlines with predicted completion times, where the former must already be specified and the latter must be estimated or derived from empirical knowledge (log extraction). They describe different types of static and dynamic prediction techniques. Additionally they differ between prediction for single-case and multi-case, where the point of view for multi-case prediction is a global one, based on the degree of utilization, assuming that high utilization will most likely result in late process instances. However, the core contribution is a comprehensive list of escalation strategies (listed and described in Section 3.6.1), which they apply and compare in different scenarios. The escalation strategies, applied in the scenarios, are hard-coded – either as inherent conditional parts of the process like "if (process late) then {perform 1 review} else {perform 3 reviews}" or as rules like "(start_D - start_B > 2 month) → add man-power".

A.2.16 Weiping Li and Yushun Fan

The technique described in [71] aims at checking the satisfiability of time constraints at build time. The authors basically duplicate the results of Eder and Panagos et al. [35] – the only extension is a time-zone constraint, which allows the specification of processes which span several time-zones. A time-zone constraint is treated like a foreseeable delay between activities, e.g. when an activity should be started at a time which is outside the working hours another time zone, it is delayed until work starts there. Their approach implies that the location, where an activity will be executed, must be known in advance (at build time). Therefore they do not support dynamic (global) role-based allocation of tasks. The model allows for sequential, parallel, and conditional structures, where the latter two are treated alike (worst case approach). As for loops, they advise encapsulating them in complex activities. Run time issues are merely mentioned.

A.2.17 Gregório Baggio, Jacques Wainer and Clarence Ellis

The authors of [4] describe a scheduling approach that aims at minimizing the number of deadline violations and the tardiness. Their prioritization technique increases the priority
of late instances, such that adhering work items will be treated in a privileged manner by workflow participants (see also Section 3.6.4.3). At first the authors examine differences between regular scheduling, like job shop and flow job scheduling, and workflow scheduling. Uncertainties, caused by conditional branches and varying execution durations, are ranked among the main challenges to overcome. Therefore they propose the use of a stochastic scheduling model. Durations are taken from a uniform distribution over a given interval, and the branching behavior is represented by estimated branching probabilities at or-splits. They do not consider parallel or cyclic structures. For each process instance (job) the start time (release date) and the deadline (due date) is known.

They compare well-known scheduling strategies like FIFO, SIRO, and EDD with their own technique, called guess-and-solve-approach (GA). GA guesses the expected execution time of each step as well as the route a job will follow, according to the given stochastic information. Based on these guesses, the problem can be reduced to a deterministic scheduling problem, which is solved by using a genetic algorithm. Additionally they describe an extension which allows dynamic scheduling by taking the current workload into consideration.

A.2.18 Ruopeng Lu and Shazia Sadiq et al.

The authors of [72] show how to apply TCNs for business process execution. They start with a discussion of existing TCN-techniques and their fundamental concepts like interval or point algebra frameworks (see also Section A.1.1). The core part of the paper deals with the definition of a business process constraint network, based on the interval algebra of James Allen [1], and how to check its satisfiability by applying a path-consistency algorithm. They also propose storing an instance template for each process instance type to capture conditional execution behavior. During run time they propose dynamically switching between instance templates, adjusting and validating them according to the current status and constraints that must still be satisfied. This approach primarily aims at checking the satisfiability of complex constraint networks during build and periodically during run time, as well as raising exceptions as soon as constraint-satisfiability can no longer be guaranteed – similar to the prediction of upcoming deadline violations. As is common practice in TCN-based techniques their model basically supports sequential and parallel structures. Furthermore conditional structures are treated by considering different instance types in isolation. Cyclic structures are not considered at all.

A.2.19 Eder and Pichler et al.

Parts of the results presented in this thesis have already been published by Eder, Pichler, and various co-authors. In [36] (based on results published in the master thesis [87]) we proposed the usage of duration histograms and branching probabilities to calculate probabilistic remaining times. [37] shows how to calculate a probabilistic timed graph,
containing E and L-histograms, which can be utilized to generate personal schedules. In [8] and [9] we extended the probabilistic model with fixed-date constraints and described how to predictively schedule processes in order to avoid unnecessary delays. In [38] and [39] we showed how to apply time management techniques on web service compositions, and explained how to deal with temporal specifics of distributed service-oriented processes. All techniques are based on a full-blocked workflow model, that allows the specification of sequential, conditional and parallel structures. For cyclic structures we proposed transforming them into a expanded structure of conditional blocks based on empirical knowledge about the iteration behavior (as described in Section 6).

A.3 Related Research Areas

Apart from literature that deals with the core problems of workflow time management, several other research areas can be identified, that either deal with tangent issues, like extracting empirical information from workflow logs, or ongoing work, like temporal issues in service-based web-environments.

A.3.1 Workflow Log Extraction and Process Mining

Workflow logs, also called workflow history, audit trails, event logs or execution logs, provide the means for gathering empirical information for performance analysis or time management purposes. Each workflow system usually provides its own proprietary log. Thus it is difficult to make general statements about log content or structure. In literature it always depends on the application scope, defined by the answers that shall be provided to workflow participants and administrators. In most cases these questions circle around time, resource and cost issues. To find answers to these questions several properties must be logged during the execution of process instances, for example: time stamps of events, which includes state changes for instance from running to suspended, as well as involved participants and other resources, costs produced and even exceptions raised [81, 52, 124, 32, 94]. These log entries are mined in order to extract the necessary information – for example processing times of specific processes and activities, transport and idle times, resource utilization, branching behavior or the system workload. Tools exist, such as ProM, which allow the extraction of process and performance information from execution logs from various workflow systems [112]. Usually commercial products like Tibco Staffware or SAP Workflow offer features to extract such information from their logs. Some approaches go even further – they deal with the identification of frequent process structures in ad-hoc workflow processes [109, 26].
A.3.2 Business Process Intelligence and Data Warehouses

Another tangent area is business process intelligence or business intelligence \[47, 43, 17\]. BI analyzes and tracks data in order to provide performance indicators for the accomplishment of business goals. It mines historical (heuristic) data of processes, stores them in specially designed data warehouses, offers features to query those warehouses. Querying and creating reports is based on online analytical processing of a multi-dimensional data warehouse, which allows spotting trends or drilling-down on certain performance measures. A very important aspect is the ease of use – handling the tools should require no or only little user input and the presented data should be intuitively understandable. However, BI does not concentrate on temporal issues, it follows a holistic approach – it gathers, interprets and reasons about every bit of data it can get hold of. Standard features may of course include temporal aspects, as for example providing turnaround times or expected remaining times based on extracted heuristics. But note that, as most systems are customizable, it is basically possible to generate every kind of information extractable from a workflow log. \[32\] proposes selecting measures like runtime-so-far, runtime-rest, finish-time, estimated finish-time, and so on. Therefore one can state that (some) predictive workflow management features should be part of a BI-enabled system. Leading manufacturers of commercial systems quickly recognized the strengths and opportunities of BI and offered corresponding tools, for instance SAP Business Intelligence with its Business Information Warehouse\(^1\) or Microsoft's SQL Server Business Intelligence features\(^2\).

A.3.3 Simulation-based Scheduling and Forecasts

Process simulation has always been a valuable tool assessing and optimizing processes, usually used during the design or re-engineering phase of a business process \[69\]. The authors of \[59\] go one step further. They argue against static scheduling approaches and recommend applying process simulation during runtime to generate schedules dynamically. They roughly outline the necessary components for such an architecture, including a simulation engine that gets the necessary information from a database utilizing data from the process history as well as from currently running process instances. A simulation-based approach has the advantage that it takes the current status, based on information about currently active process instances and predicted future work load, into consideration. Simulation-based prediction has already been integrated in commercial workflow products – for example the Tibco Staffware Process Suite allows forecasting upcoming workflow steps as well as workload estimations \[101\].

\(^1\)www.sap.com
\(^2\)www.microsoft.com
A.3.4 Temporal Concepts in Service-based Applications

Nowadays business processes spread over the boundaries of companies and integrate customers, suppliers and partners in order to achieve inter-organizational business goals. Inter-organizational business processes are therefore assembled from external processes and services [2]. Inter-organizational business processes can be very dynamic, as services forming the process may be selected from a number of alternatives offered by different providers. To assess the quality of a service it is necessary to define measures which are significant indicators for certain quality aspects, where expected or guaranteed process duration ranks among the most important characteristics [16]. To guarantee a certain quality partners even define service level agreements [46]. An SLA precisely defines quality measures like response times along with the according costs and specifies the penalties if the provider is not able to meet the contract. As with regular workflows a slow service can have a disastrous impact on the overall process response time and even worse result in the violation of time constraints. Thus techniques are needed to predict these durations and possible constraint violations based on the anticipated response time of participating web services, and to exchange certain services or to optimize and schedule them for faster execution (e.g. [95, 5, 39]).

A.4 Systems

Although the field of time management has received a lot of attention in areas such as project management, job-shop scheduling, and active databases, most currently available workflow products provide little support beyond simple monitoring activity deadlines. Many commercial workflow systems are still 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 of these conditions is supported [3]. Systems with such limited time management capabilities as well as systems which provide predictive features based on the integration of a business intelligence component are not discussed here (see also Section A.3.2).

A.4.1 Adept

In [24, 25, 89] the authors outline some temporal extensions to the ADEPT workflow management system (research prototype), called ADEPT_time. As part of the time functionality, minimal and maximal durations may be specified for each activity, as well as deadlines and time dependencies between activities, similar to upper bounds constraints presented in Section 3.4.2.2. According to [89] they applied temporal constraint networks for representing and checking time constraints and that during build time the satisfiability of constraints is verified. However, details remain unclear – they do not show how TCN-concepts are mapped on their version of a workflow graph, which allows full-blocked
modelling of sequential, parallel, conditional and cyclic structures, plus some even more advanced control flow structures. They additionally claim that during run time ADEPT informs users when deadlines are going to be missed – which implies forecasting capabilities and predictive features! Again details remain unclear, for instance how they deal with conditional structures or loops, how run time mapping and prediction works, and so on.

A.4.2 Phanta Rhei

In [27] the authors describe, among other things, the integration of time management features into their web-based workflow system Phanta Rhei (research prototype). Their time management component utilizes the ePERT-approach [88], which forms the basis for the extended probabilistic model presented in this thesis. The prototype solely implements reactive and predictive features, like detecting late processes and raising corresponding exceptions – proactive features have not been integrated. For further details on ePERT please refer to Chapter 3 and according related work in Section A.2.

A.4.3 Tibco Staffware

The last version of the Tibco Staffware Process Suite [100] provides some features which are very similar to those provided by workflow time management. Their prediction component utilizes historical data and the control flow structure, in order to forecast future task assignments, as well as predict deadline violations. As already mentioned above, they additionally offer forecasting based on live case data and simulation, called on-demand prediction. However, until now they do not offer automated proactive features – they propose using the predicted information to optimize the process. Although the objectives and features are very similar to those of predictive time management described in this thesis, they applied, according to [14], a fundamentally different technique: complex event processing. CEP basically utilizes (empirical or simulated) knowledge about the causality of events, determines how these events affect critical functionality in eventually constrained systems, and generates event patterns to be applied in live systems. For details on this topic please refer to [73].
Bibliography


[22] Compaq. Compaq Team Links Mail V4.5 - Software Product Description. SPD 63.25.06.


