Marco Franceschetti

Temporal Variables for Business Process Modeling

DISSERTATION

submitted in fulfilment of the requirements for the degree of

Doktor der Technischen Wissenschaften

Alpen-Adria-Universität Klagenfurt

Fakultät für Technische Wissenschaften

1. Supervisor: O.Univ.-Prof. DI Dr. Johann Eder

Universität Klagenfurt / Institut für Informatik Systeme

2. Supervisor: Prof. Dr. Carlo Combi

Università degli Studi di Verona / Dipartimento di Informatica

Evaluator: Prof. PhD Barbara Weber

Universität St. Gallen / Institute of Computer Science

Evaluator: Prof. Marcello La Rosa

The University of Melbourne / School of Computing and Information Systems

Klagenfurt, July/2020

Affidavit

I hereby declare in lieu of an oath that

- the submitted academic paper is entirely my own work and that no auxiliary materials have been used other than those indicated,
- I have fully disclosed all assistance received from third parties during the process of writing the thesis, including any significant advice from supervisors,
- any contents taken from the works of third parties or my own works that have been included either literally or in spirit have been appropriately marked and the respective source of the information has been clearly identified with precise bibliographical references (e.g. in footnotes),
- to date, I have not submitted this paper to an examining authority either in Austria or abroad and that
- when passing on copies of the academic thesis (e.g. in bound, printed or digital form), I will ensure that each copy is fully consistent with the submitted digital version.

I understand that the digital version of the academic thesis submitted will be used for the purpose of conducting a plagiarism assessment.

I am aware that a declaration contrary to the facts will have legal consequences.

Marco Franceschetti, m.p. Klagenfurt, 2020 Juli 12

Acknowledgements

I am genuinely grateful to my supervisor Prof. Johann Eder for his guidance through my doctoral journey. His encouragement and support have been essential to make this work possible. I am also grateful to my co-supervisor Prof. Carlo Combi for offering his expertise to support my work. My thanks go also to Prof. Barbara Weber and Prof. Marcello La Rosa for reviewing this thesis.

I also would like to thank Prof. Roberto Posenato, for his inputs during several discussions around Temporal Constraint Networks. He provided invaluable insights in the formalisms, which have been essential for the progress of my research efforts.

I wish to thank also Prof. Julius Köpke, for numerous discussions, suggestions, and encouragement throughout my doctoral studies, as well as for the fruitful research collaboration, which led to a number of publications together.

Many thanks go to my colleagues and friends at the Department of Informatics Systems, for engaging in stimulating discussions, as well as for providing support in the various academic activities.

I would also like to thank my "extended IEEE" friends in Klagenfurt: Giovanni, Giacomo, Petra, and Alice. With them it was possible to find the necessary distraction, and enjoy the local gastronomy. Of course I also thank my friends in Italy, who supported me during these years.

I am very grateful to my parents and my brother, who always supported me. Without them, I would have not been able to pursue my doctoral studies.

Finally, I wish to thank Holly, for her constant support, love, and patience.

Abstract

Business Process Management is a technical discipline which has an increasingly significant impact on the way organizations carry out their work, and create value, thanks to automation. The support of information systems to a faithful modeling of business processes is essential for making process automation beneficial to stakeholders.

Among the several perspectives of a process, the temporal perspective needs particular care. Capturing all temporal requirements, guaranteeing timeliness of response and operation, finding agreements around which temporal guarantees can be achieved in a collaboration, are all tasks which process designers need to cherish. Poor time management in business processes, starting from the design of temporal requirements of a process, may lead to a number of consequences, which range from mildly inconvenient to fatal.

A number of shortcomings can be observed in the current design time support to time management for process designers. These shortcomings include inadequate modeling language expressive power, insufficient enforcement of business secrecy, and lack of consensus on a criterion for considering process models to be temporally correct and thus qualified for being executed.

With this work we take the challenge of increasing the expressive power of modeling languages for business processes, by introducing temporal variables. Temporal variables are data elements which allow the expression of temporal information. We show how temporal variables allow process designers to overcome shortcomings of current modeling languages, such as inadequate expressive power for the definition of a wide range of temporal constraints, and lack of support to business secrecy in time-constrained cross-organizational processes. We also show how, by leveraging on existing formalisms for temporal reasoning, process designers can be assisted in verifying the temporal correctness of their models, and in computing missing temporal information holding temporal correctness guarantees.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.1.1 Modeling Temporal Requirements . . . 2

1.1.2 Verifying Temporal Correctness . . . 3

1.2 Aims . . . 4

1.3 Contributions . . . 4

1.4 Research Methods . . . 5

1.5 Structure . . . 5

1.6 Published Work . . . 7

2 Background **9**

2.1 Business Process Technology . . . 10

2.1.1 Historical Aspects of Business Process Technology . . . 10

2.1.2 Business Process Concepts . . . 10

2.1.3 Terminology . . . 12

2.1.4 The WfMC Reference Architecture . . . 13

2.2 Process Design . . . 16

2.2.1 Aspects of Process Models . . . 16

2.2.2 Process Modeling Languages . . . 17

2.3 Temporal Constraint Networks . . . 21

2.3.1 Temporal Constraint Network Types . . . 21

2.3.2 Controllabilities . . . 27

2.3.3 Checking Dynamic Controllability . . . 28

3 Problem Description and Research Questions **33**

3.1 Problem Description . . . 33

3.1.1 Dependency on Temporal Data . . . 34

3.1.2 Temporal Constraints Across Modules 36
3.1.3 Temporal Constraints Across Distributed Processes 37
3.2 Research Questions 37

4 Process Models with Temporal Variables 43
4.1 Introduction 43
4.2 Architecture, Context, and Assumptions 43
4.2.1 Task Durations 44
4.2.2 Temporal Constraints 45
4.2.3 Temporal Variables 46
4.3 Intra-Organizational Processes 47
4.3.1 An Example for an Intra-Organizational Process 47
4.3.2 Intra-Organizational Process Model with Temporal Variables . . 49
4.3.3 Modularized Processes 51
4.4 Web Service Compositions 52
4.4.1 An Example for a Web Service Composition 53
4.4.2 Process Model for Web Service Compositions with Temporal Parameters 53
4.5 Cross-Organizational Processes 55
4.5.1 An Example for a Cross-Organizational Process with Temporal Parameters 56
4.5.2 Local Process Model in a Cross-Organizational Process 57
4.6 Temporal Patterns and Applications 59
4.6.1 Dependency on External Data 60
4.6.2 Modularized Processes, Web Service Compositions, and Cross-Organizational Processes 61
4.7 Temporal Semantics 62
4.8 Conclusion 64

5 Checking Temporal Correctness 65
5.1 Introduction 65
5.2 Motivating Dynamic Controllability 65
5.2.1 How to Check Dynamic Controllability 66
5.3 Mapping Process Definitions into Temporal Constraint Networks 67
5.3.1 Mapping Processes into STNUs 68
5.3.2 Mapping Processes into CSTNUs 70
5.4 Checking Dynamic Controllability 71

5.5 Implementation 73
5.6 Conclusion 75

6 Computing Restrictions for Temporal Parameters 77
6.1 Introduction 77
6.2 Computing Input Temporal Parameter Restrictions 78
6.3 Constraint Propagation Rules 80
6.4 An Algorithm to Derive Parameter Restrictions 84
6.4.1 Interpretation of the Results 84
6.4.2 Soundness and Completeness of the Algorithm 85
6.5 Implementation 87
6.6 An Alternative Approach to Compute Missing Ranges 88
6.6.1 A Necessary and Sufficient Range for an Input Parameter 89
6.6.2 Computing Ranges for Input Parameters 90
6.6.3 Implementation and Evaluation 91
6.7 Comparison of the Two Approaches 92
6.8 Conclusion 93

7 Negotiating Restrictions for Temporal Parameters 95
7.1 Introduction 95
7.2 Cross-Organizational Process Model 97
7.3 Temporal Commitments 98
7.4 Negotiating Parameter Restrictions 99
7.5 A Procedure for Negotiating Constraints 101
7.5.1 Interpretation of the Results 102
7.5.2 Soundness and Completeness of the Procedure 102
7.6 Alternative Negotiation Approaches 104
7.7 Conclusion 104

8 Related Work on Time Management in Business Processes 107
8.1 Introduction 107
8.2 Time Management in Business Processes: Overview and Challenges . . 108
8.2.1 The Temporal Perspective in Business Process Modeling: A Survey and Research Challenges [23] 108
8.2.2 Workflow Time Management Revisited [58] 109
8.2.3 Process Time Patterns: A Formal Foundation [96] 110

8.3 Modeling, Management, and Verification of Temporal Aspects: A Historical Overview . . . 111
8.3.1 C. Bussler (1998), and L. J. Zhao and E. A. Stohr (1999) . . . 111
8.3.2 O. Marjanovic et al. (1999-2000) . . . 112
8.3.3 J. Eder et al. (1999-2000) . . . 112
8.3.4 E. Kafeza and K. Karlapalem (2000) . . . 113
8.3.5 H. Zhuge et al. (2000-2001) . . . 113
8.3.6 C. Bettini et al. (2000-2002) . . . 113
8.3.7 C. Combi and G. Pozzi (2003-2004), and C. Combi et al. (2007) . 114
8.3.8 R. Lu et al. (2006) . . . 115
8.3.9 J. Eder and A. Tahamtan et al. (2007-2012) . . . 115
8.3.10 C. Combi et al. (2009-2019) . . . 116
8.3.11 A. Lanz et al. (2013-2019) . . . 117
8.3.12 J. Eder and M. Franceschetti et al. (2018-2020) . . . 117
8.4 Other Work on Modeling and Reasoning about Time in Business Processes 118
8.4.1 Temporal Logic and Event Calculus . . . 118
8.4.2 Dynamic Condition Response Graphs . . . 119
8.4.3 Modeling Durations and Temporal Constraints in BPMN . . . 119
8.4.4 Process Mining . . . 120
8.4.5 Compliance Checking . . . 120
8.4.6 Time in Service Level Agreements . . . 121

9 Conclusions **123**
9.1 Conclusion . . . 123
9.2 Answers to the Research Questions . . . 126
9.3 Outlook . . . 128

List of Figures

2.1 Relationships between basic terminology. Image from [27]. 14
2.2 Components and interfaces of the Workflow Reference Model. Image from [78]. 15
2.3 Example for a Petri net. 18
2.4 Example for a Workflow graph from [53]. 19
2.5 Example for a Business Process Diagram for a cross-organizational process. 20
2.6 Example for a Simple Temporal Network with Uncertainty (STNU). . . 24
2.7 Example for a Conditional Simple Temporal Network with Uncertainty (CSTNU). 26
2.8 Constraint propagation rules for STNUs developed by Morris and Muscettola. Image from [17]. 31
2.9 Constraint propagation rules for STNUs developed by Cairo and Rizzi. Image from [17]. 32

3.1 BPMN-like fragment of a medical process with temporal constraints. . . 34
3.2 BPMN-like fragment of a modularized process with temporal constraints. 36
3.3 BPMN-like fragment of a cross-organizational process with temporal constraints. 38

4.1 Medical process with time information treated as data. 48
4.2 Example Web service composition with temporal parameters. 53
4.3 Example process in BPMN-like notation with cross-organizational constraints. 56
4.4 Modularized process with module time information treated as data. . . 62

5.1 Mapping rules to map a process into a STNU. 69

5.2 STNU equivalent to the process of Figure 4.1 obtained with the application of the mapping rules of Def. 5.3.1. The graphical notation follows the notation of [116]. For a compact presentation, the initials of the process nodes are used for labeling the corresponding STNU nodes. . . 70
5.3 Mapping rules to map a process into a CSTNU. 72
5.4 Mapping application architecture pipelined with STNU controllability check tool. 73

6.1 Overview of the approach for computing parameter restrictions. 80
6.2 P-Relax rule. The derived constraint is dotted. 81
6.3 P-Lower rule. The derived constraint is dotted. 82
6.4 P-Upper rule. The derived constraint is dotted. 83
6.5 Restrictions for the parameters of the Web service composition of Fig. 4.2 as STNU constraints, obtained from the application of Alg. 6.1. 86

Chapter 1

Introduction

1.1 Motivation

A structured approach to the design and execution of business processes has been put in practice in the last decades, with indubitable benefits in the way companies and organizations accomplish their goals and create value. Business Process Management (BPM) and process automation, with the adoption of workflow technology, Enterprise Resource Planning (ERP) systems, database systems, and other IT-driven solutions, are key factors for the success of the process-oriented paradigm. It is expected that the next decades will witness an increasingly pervasive adoption of such automated solutions [74, 148].

A particular benefit of automation resides in the automatic management of process requirements, instead of delegating it to humans. Automation in the management of requirements offers advantages, such as reduced quality fluctuations, and deterministic response during process execution. To enable automation it is necessary to adequately model, in a process specification, all its requirements. In particular, temporal requirements are a crucial part of a process specification. Failing to meet a temporal requirement may lead to the violation of a contract, of an obligation, of an expectation, or even of the law. The consequences of not meeting a temporal requirement include reduced trust, high compensation costs, and potential cascading failures [58].

1.1.1 Modeling Temporal Requirements

Temporal requirements is a broad term, which translates to a number of process elements. These elements are related to the temporal dimension of a process, and include in particular durations, temporal constraints, deadlines. A duration refers to the time span required for the execution of some process task. A temporal constraint reflects the requirement that a certain amount of time is allowed to elapse between two events during the execution of a process. Deadlines, instead, specify points in time which represent a temporal limit for the occurrence of events.

Temporal constraints are requirements, which state that certain time spans must be ensured between two given events. We consider here *upper-bound* and *lower-bound* constraints. Upper-bound constraints require a maximum lapse of time between two events; conversely, lower-bound constraints require a minimum time lapse between two events. So for instance, an upper-bound constraint may require that communicating the result of a job application must be done within one week after the decision about the chosen candidate was taken. Which are the events, on which such temporal constraints can be imposed? Current process modeling languages allow the formulation of temporal constraints which refer to process events only. These are events which are bound to some control flow elements, such as the start of the process or of an activity, or its end. However, there are real world scenarios in which also other events, such as dates on documents, or events occurring at other processes, are used to establish some temporal constraint. An example for such a requirement is the restriction of validity for a one-time password by giving a deadline, which may be communicated as a timestamp, after which a login procedure using that password cannot be executed. Of course it would be desirable for process designers, to be able to model business processes which include also temporal requirements referring to events which are not bound to any process task.

In addition to the need to set temporal constraints on events which are observable within a process, but do not refer to any process event, is the problem of setting temporal constraints in a cross-organizational setting. In cross-organizational processes, a number of parties locally execute their processes in coordination with each other. A temporal constraint in such a cross-organizational process is used to establish a restriction between events occurring at two different processes. This requires that the involved parties have access to each other's local process implementation in order to enforce the temporal constraint. However, at the same time, each collaborating party

typically requires that no information regarding its local process implementation is made public. This demand for secrecy is recognized to be in contrast with the necessity to establish temporal constraints which span across multiple processes in the collaboration [60].

1.1.2 Verifying Temporal Correctness

In recent years there has been increasing consensus that real world processes distinguish between durations of two different types, which are inherent to the nature of the task they are associated to. These two duration types are known as *non-contingent* and *contingent*. A non-contingent duration is associated to a task which is under full control of the executor, who can decide for how long the task is carried on, as long as this duration is within a specified interval. In contrast, a contingent duration is associated to a task which is not under full control of the executor. In such a case, the duration is merely observed, but not controlled. Examples for contingent durations are durations for bank money transfers, or for chemical reactions in environments whose physical-chemical setting cannot be controlled [2, 51].

The coexistence of contingent and non-contingent durations in processes makes the scheduling of these processes quite difficult, since contingencies introduce a degree of uncertainty, which makes the run time behavior of a process less predictable. Over the years, however, the temporal problem associated to verifying the existence of a solution for a set of temporal constraints and durations, some of which are not controllable, has fostered numerous research efforts. The result is a number of elaborated notions for temporal correctness, formalisms, and techniques, which make it possible to know whether it is possible to compute schedules, which allow the fulfillment of any existing temporal constraint, despite uncertainties.

Towards the modeling and verification of temporal problems, extensive research has been conducted in the last years, in particular around the formalism of Temporal Constraint Networks (TCNs). TCNs have been proposed to support the solution to the Simple Temporal Problem (STP) in [45]. Especially research from the Artificial Intelligence community has produced further refinements of these temporal networks, such as extensions for representing contingent durations [115] or conditional executions [136].

In parallel with the refinement of the modeling capabilities of TCNs, sophisticated notions for temporal correctness were further developed. Observing these advance-

ments, the Business Process Management community started considering these works, developing methods based on temporal networks for verifying the temporal qualities of business process models, such as in [36]. Thus, resorting to TCNs is now a quite established approach for the verification of the temporal correctness of time-constrained process models.

1.2 Aims

With this work we want to cater to the need to establish temporal constraints on events which are not restricted to the control flow of a single process. Instead, we aim at providing process designers with an extended modeling language, which allows them to refer to a broader set of events. These events include time points stamped in artifacts manipulated by process tasks, and events occurring in other communicating processes.

Extending the languages for process modeling as we intend to do here, requires taking into account the need of process designers to verify the temporal correctness of their process models. Of course, this means that, in the light of the advancements in the theory, we need to consider the most refined notion for temporal correctness. Currently, such a notion is known as *dynamic controllability*. Thus, with this work we also aim at empowering process designers with methods to check the dynamic controllability of process models, which are built around our proposed language.

1.3 Contributions

The contributions of this work are the following:

- A process model featuring temporal variables for representing events in time, which allows the definition of a number of different process types: intra-organizational processes, modularized processes, Web service compositions, and cross-organizational processes.
- Mapping rules to map a process definition with temporal variables into a temporal constraint network, for checking its dynamic controllability.
- A set of rules to derive restrictions on temporal variables of a process definition, which are independent of any contingencies in the process definition.

- A procedure based on these rules to derive restrictions on temporal variables, which are necessary conditions to fulfil, in order to guarantee the dynamic controllability of a process definition.
- A distributed procedure for the negotiation of restrictions on temporal variables in cross-organizational processes, which guarantee the dynamic controllability of all participating local processes.

1.4 Research Methods

The approach we take for solving the problems we target in this thesis is based on design science methods, as described by Wieringa in [149].

In particular, in this thesis we provide a number of artifacts in the form of:

- **Formal definitions** for the formalization of concepts. We give definitions for various process models, to represent different types of business processes; we formally define their semantics; we define mapping rules to project the temporal features of a process model into an equivalent TCN; we define rules for inferring temporal knowledge.
- **Lemmas, Theorems, and Proofs** for stating formal properties and substantiating them. We formalize as theorems and prove the correctness of a number of observations on our definitions.
- **Algorithms** for defining procedures. We define algorithms for checking the temporal correctness of process models, for deriving new temporal knowledge, and for computing agreements on temporal commitments.
- **Implementation of algorithms** for evaluating their feasibility and scalability. We implement the proposed algorithms, and evaluate their behavioral properties.

1.5 Structure

The remainder of this thesis is structured as follows:

- In **Chapter 2** we give an overview of the background concepts in business process technology, and of the modeling and reasoning capabilities offered by Temporal

Constraint Networks. The goal of this chapter is to provide an explanation of the essential concepts at the basis of this work.

- In **Chapter 3** we emphasize the limitations affecting current modeling languages for business processes in relation to the expression of temporal requirements, and to the verification of temporal qualities of business processes. We use these limitations as the motivation for the research questions driving this work, which are also presented in this chapter.

- **Chapter 4** introduces a new process model for the expression of business processes, which features temporal variables as a means to model events which current languages are not able to capture. We introduce a basic process model for the expression of intra-organizational processes, and present extensions to it, which allow us to capture also Web service compositions and cross-organizational processes. In this chapter we additionally give a formalization of the semantics of such process model, and show, through the means of examples, the temporal patterns enabled by the introduction of temporal variables.

- **Chapter 5** digs into the problem of checking the temporal correctness of process definitions, which are based on our proposed process model. We motivate dynamic controllability as the notion which process designers should strive for, and define techniques, based on Temporal Constraint Networks, which can be used for checking the dynamic controllability of process definitions which have a complete specification of their temporal requirements.

- In **Chapter 6** we show how Temporal Constraint Networks may be used to define techniques for computing missing temporal restrictions for process models, such that their fulfilment gives guarantees of temporal correctness in the sense of dynamic controllability. We show how these techniques may be used by service providers in the context of Web service compositions, to compute Temporal Service Level Agreements.

- In **Chapter 7** we delve into the setting of cross-organizational business processes. We show how our process model with temporal variables enables a secrecy-oriented design of business collaborations, which require temporal coordination. We stress the need for distributed procedures which compute temporal requirements agreed by all the parties in a collaboration. We propose a procedure for the negotiation of temporal requirements on the temporal parameters exchanged in a cross-organizational business process.

- **Chapter 8** discusses a number of research efforts which we see at the basis of our contribution, showing how incremental efforts by a number of authors posed the ground, which made this work possible. We also take a look at other research contributions which are relevant in the domain of time management for business processes.
- **Chapter 9** summarizes this thesis, discussing the problems which remain open, giving an outlook on future work.

1.6 Published Work

Part of the contents of this thesis have already been published in the scientific literature. Such publications are the following:

- Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of processes with temporal variables. In Chih-Cheng Hung and George A. Papadopoulos, editors, *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019*, pages 40-47. ACM, 2019. [51]
- Marco Franceschetti and Johann Eder. Checking temporal service level agreements for web service compositions with temporal parameters. In Elisa Bertino, Carl K. Chang, Peter Chen, Ernesto Damiani, Michael Goul, and Katsunori Oyama, editors, *2019 IEEE International Conference on Web Services, ICWS 2019, Milan, Italy, July 8-13, 2019*, pages 443-445. IEEE, 2019. [62]
- Marco Franceschetti and Johann Eder. Computing ranges for temporal parameters of composed web services. In *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services, iiWAS 2019, Munich, Germany, December 2-4, 2019*, pages 537-545. ACM, 2019. [63]
- Marco Franceschetti and Johann Eder. Dynamic service binding for time-aware service compositions. In *23rd IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2019, Paris, France, October 28-31, 2019*, pages 146-151. IEEE, 2019. [64]
- Marco Franceschetti and Johann Eder. Computing admissible temporal slas for web service compositions. In *2020 IEEE International Conference on Web Services, ICWS 2020, Beijing, China, October 20-23, 2020 (forthcoming)*, 2020. [66]

- Marco Franceschetti and Johann Eder. Designing decentralized business processes with temporal constraints. In *CAiSE Forum 2020, Held at CAiSE 2020, Grenoble, France, June 8-9, 2020 (forthcoming)*, 2020. [67]
- Marco Franceschetti and Johann Eder. Negotiating temporal commitments in cross-organizational business processes. In *27th International Symposium on Temporal Representation and Reasoning, TIME 2020, Bolzano - Bozen, Italy, September 23-25, 2020 (forthcoming)*, 2020. [68]

Chapter 2

Background

The widespread adoption of Business Process Technology which took place in the last decades at both enterprise and organizational level has been an enabling factor for increased efficiency, and operational cost reduction, and it facilitated the cooperation between different entities. However, despite the numerous challenges which have been addressed to make Business Process Technology possible and usable, there are still challenges which remain unaddressed.

Contextually, significant advancements have been accomplished in the research around time-related problems in various application areas, such as Artificial Intelligence, planning, and scheduling. In particular, in the last three decades, the formalism of Temporal Constraint Networks has seen increasing attention from researchers. These research efforts have produced important results, such as increased expressiveness, and effective algorithms to provide answers to questions around the temporal correctness of such networks. We consider it as a prime time now, for Temporal Constraint Networks, to become the formal framework for addressing the temporal problems in the field of Business Process Technology.

In this chapter we discuss the current state of the art in the two streams of research of Business Process Technology and Temporal Constraint Networks, as a basis for the following discussion, which we present in the next chapters. We first give an overview of Business Process Technology, provide definitions for the concepts and terms in the area of business processes, and discuss the major challenges and areas of research which have been addressed. We then give an overview on the existing work in the area of Temporal Constraint Networks, upon which we will build part of our contribution.

2.1 Business Process Technology

The first stream of research at the basis of this thesis is the work around the IT support to Business Process Management, in particular the work on process design. We introduce here the most important concepts in the area, repeat the most important term definitions, and give an overview on the reference architecture as defined by the Workflow Management Coalition.

2.1.1 Historical Aspects of Business Process Technology

Business Process Technology can be dated back to the 1980s, with the adoption of solutions for office automation, database management, and electronic communications. One of the novelties of Business Process Technology as intended today, however, is the separation of the process logic form the application logic in a work environment which is both technical and process oriented [148]. Often cited advantages of the interplay of structure and automation include clear definition of responsibilities, transparency over the progress of execution, predictability of execution times, reduced communication costs and times, support to systems interoperability through defined interfaces.

The growing interest and increasing adoption of Business Process Technology led, in 1993, to the founding of the Workflow Management Coalition (*WfMC*). The WfMC is an international organization which puts together vendors, analysts, users, and researchers of Business Process Technologies, with the initial goal of harmonizing and making inter-operable different workflow systems. The work of the WfMC resulted, among other contributions, in the definition of a glossary, and the standardization of workflow system architectures, interfaces, and process definition languages. We give an overview of these results in the next sections.

2.1.2 Business Process Concepts

The very first concept which needs to be defined is that of business process. Weske states that *a business process consists of a set of activities that are performed in coordination in an organizational and technical environment* [148]. This definition captures the nature of close interplay between human and technology which is crucial for increased efficiency of business processes, as opposed to a traditional, unstructured way of organizing work.

Due to the widespread adoption of Business Process Technology, a whole discipline of *Business Process Management* (*BPM*) developed around the concept of business process. With BPM we refer to the formalization of concepts and terms, and to the techniques supporting the *business process lifecycle*.

The business process lifecycle includes the phases of *process identification*, *process discovery*, *process analysis*, *process redesign*, *process implementation*, and *process monitoring and controlling*. We give a brief overview of the lifecycle phases here. A much deeper insight can be found in [49]. In the process identification phase, the aim is identifying a relevant business problem and the processes which exist around it. With the process discovery phase, these processes are documented with the use of some modeling language. A wide number of notations exists for this purpose, the most common of which will be presented in Sect. 2.2. As pointed out in [49], this phase is frequently referred to as *process design*, even though the argument against such a term is that in reality processes already exist, at least in some abstract form, therefore a notion of creating something *ex nihil* is not appropriate. The phase of process analysis is aimed at the identification, and possibly prioritization, of a set of issues which are associated to a discovered process model. Such issues are addressed in the process redesign phase, which is in fact intertwined with the process analysis phase. Process implementation is the phase in which the changes to apply to an existing process are put in place to transform it into a redesigned process. This phase may involve both organizational aspects, as well as technological ones, in particular the development and deployment of process automation. During the enactment of implemented process instances, measurement of metrics and generation of logs take place. This information is used in the monitoring and controlling phase by applying several techniques, such as process mining, which aim at assessing the quality of a process and identifying potential bottlenecks, as well as improvements. Decisions stemming from the monitoring and controlling phase may trigger new iterations of the lifecycle, for a continuous improvement of business processes.

From the business process lifecycle, an important distinction emerges between two distinct macro phases in the life of a process: *design time*, and *run time*. Design time comprises the phases of process identification, analysis, redesign, and implementation. Run time is the phase in which a process is put into action and is being executed. Depending on the macro phase considered, different operations may be performed on a process model for evaluating it: for instance, at design time certain temporal reasoning procedures may be applied to discover the temporal qualities of a process

model without executing it, but with the advantage of avoiding potential real-world consequences caused by its execution. On the other hand, run time based operations such as process mining may take advantage of real data generated by the execution of a process model, which reflect its actual characteristics.

2.1.3 Terminology

The WfMC, in its glossary fist published in 1994 and subsequently revised [27], gives technical definitions for concepts and terms which are used in the context of Workflow technology, which here we call Business Process Technology. The declared intent for producing such a glossary was fostering the adoption of a consistent language among the industry. The glossary was consistently adopted also by researchers; in recent years, however, the scientific literature started increasingly using synonyms for some of the terms. Some of these synonyms will be used throughout this work for consistency with this trend. Therefore, we report here the original definitions which are the most relevant for a proper understanding of the rest of this dissertation, together with their now more commonly used synonyms. Verbatim citations from [27] are in *italics*.

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

- **Process Definition** *The representation of a business process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system.The process definition consists of a network of activities and their relationships,criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.*

  Alternatively, we often refer to a process definition as a *process model* or *process implementation*.

- **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.*

  With a certain degree of flexibility, we use the term *process* as an alternative way to refer to a workflow.

- **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.*

- **Activity** *A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. [...]*

  As synonyms for activity, we also use the terms *step* and *node*.

- **Instance** *The representation of a single enactment of a process, or activity within a process, including its associated data. Each instance represents a separate thread of execution of the process or activity, which may be controlled independently and will have its own internal state and externally visible identity, which may be used as a handle, for example, to record or retrieve audit data relating to the individual enactment.*

  We often refer to an instance as a *trace*.

- **Participant** *A resource which performs the work represented by a workflow activity instance. [...]*

  Often the term *actor* is used instead of participant.

The WfMC provides an illustration which emphasizes the relationships between the concepts listed above. We report it in Fig. 2.1.

2.1.4 The WfMC Reference Architecture

The definition of a reference architecture for a Workflow Management System is a major result of the WfMC efforts to standardize Business Process Technology. It provides a guideline for the design of a workflow management system, characterized by clear separation of concerns and roles and responsibilities. The schema for the WfMC *Workflow Reference Architecture* is depicted in Fig. 2.2; in the following we give an overview of each architecture element and interface. More details can be found in [78] and in the individual interface specifications provided by the WfMC [1].

The central element of the architecture is the *workflow enactment service*, which contains one or more workflow engines. The enactment service is the component respon-

[1] https://www.wfmc.org/resources

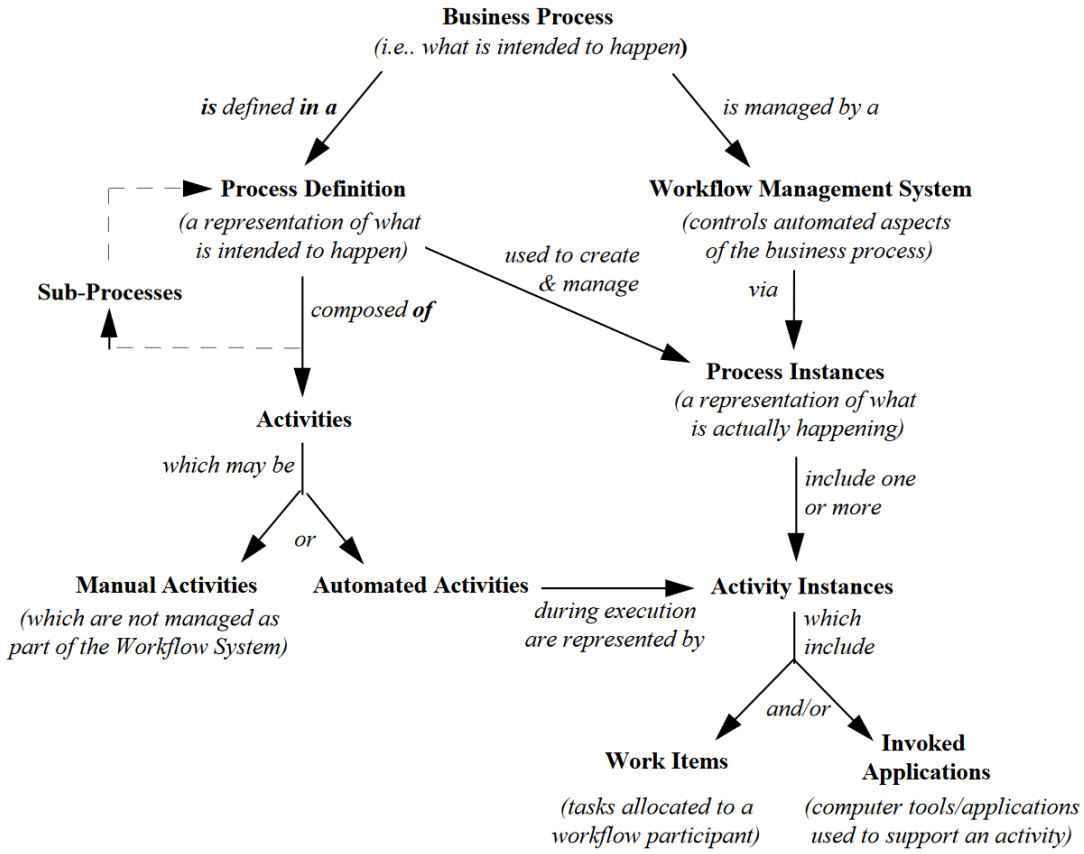

Figure 2.1: Relationships between basic terminology. Image from [27].

sible for the execution of process instances. Process instances are created after process models stored in a workflow model repository. The workflow enactment service interacts with other subsystems through a number of interfaces, each with a specific role and goals. These interfaces are defined as follows:

- *Interface 1 - Process Definition* connects the enactment service to tools for the definition of process models. The goal is to make it possible for any modeling tool developed by any vendor to make the defined processes executable by the engine.

- *Interface 2 - Client Applications* is used for connecting client applications enabling the interaction of human agents during the execution of a process instance. The

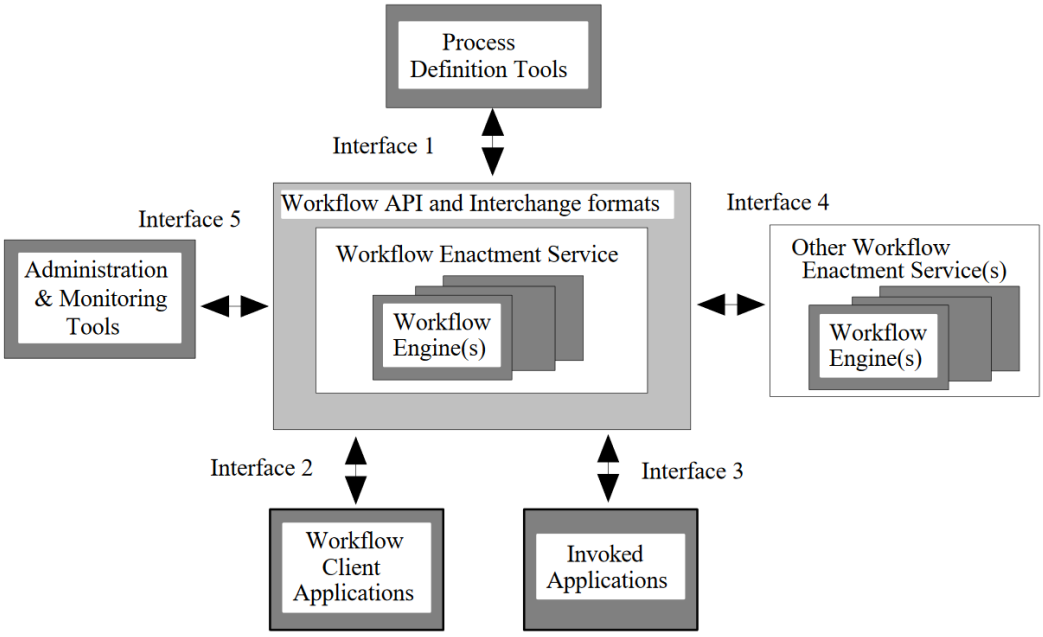

Figure 2.2: Components and interfaces of the Workflow Reference Model. Image from [78].

goal is to let any client application access the workflow engine to present users the work items they are responsible for, and support them in completing their tasks.

- *Interface 3 - Invoked Applications* supports the invocation of external applications which may be needed for task completion. The goal is to provide users an integration with a heterogeneous set of applications from within the WfMS.

- *Interface 4 - Other Enactment Services* allows for the interoperability across different workflow engines to complete a workflow instance. The goal is system interoperability, in particular across different organizations, which may adopt different workflow engines.

- *Interface 5 - Administration & Monitoring* connects administration and monitoring tools for assessing the status at runtime of process instances and collecting measurements. The goal of this interface is the potential optimization of processes and organizational administration.

We set our focus in particular on the first interface, for the definition of process models. In the next section we devote our attention to the design of business processes, giving an overview of the most widely adopted languages for defining process models.

2.2 Process Design

At the early stages of the process lifecycle is the phase of process design. Process design begins after the requirements collection phase. We focus here on the production of a model for a process, in particular on the process aspects which need to be modeled, and the most widely adopted languages, which allow the representation of these aspects.

2.2.1 Aspects of Process Models

It is recognized that process designers, in order to produce faithful process models, have to consider different aspects of their processes. We briefly discuss these aspects here, highlighting the importance of an aspect which is often not sufficiently addressed: the temporal aspect.

- **Functional Aspect:** The functional aspect of a process regards the tasks which compose the process, realizing the required functionalities.
- **Behavioral Aspect:** The behavioral aspect deals with defining the order in which the tasks are executed, which is known as the process control flow.
- **Informational Aspect:** The informational aspect is concerned with the work items which are manipulated by the process tasks.
- **Organizational Aspect:** The organizational aspect links the tasks, the control flow, and the information flow with the structure of the organization which executes the process.
- **Operational Aspect:** The operational aspects involves the IT systems and technologies required for the execution of automated or semi-automated tasks, and the physical resources needed for the non-automated tasks.
- **Temporal Aspect:** While it is often seen as a facet of the behavioral aspect, since it strictly influences the execution of the tasks, we explicitly advocate for

considering the temporal aspect as one of the main aspects, which need to be taken into account when modeling a process. In the temporal aspect we include the definition of task durations, deadlines, and of temporal constraints which restrict the times of occurrence of events, such as the start of a task execution.

In the following we discuss how these aspects can be modeled through some of the most widely adopted process modeling languages.

2.2.2 Process Modeling Languages

When it comes to the task of creating a business process model, the first question is *which language to use* for representing such process model. Similar to programming, a variety of paradigms exists to approach the modeling phase. For instance, a declarative paradigm sets the focus on the sequence of different states which characterize the execution of a process from start to end. On the other hand, an imperative paradigm defines a process in a normative manner, e.g., by stating the sequence of actions to be executed. For each paradigm several modeling languages exist, each of these languages being focused in particular on one or more specific aspects. We briefly present an overview of the most common languages, highlighting the aspects of process models they focus on.

Petri Nets and Workflow Nets

Petri nets are a formalism introduced by Carl Adam Petri in [121]. Petri nets enable the expression of behavioral properties of systems, in the form of state changes.

The building blocks of Petri nets are places, transitions, tokens, and arcs. Tokens can be distributed on, and moved between, places. The placement of tokens in the places of a Petri net determines its *configuration*, which is the representation of a state. Arcs from places to transitions, and from transitions to places, determine the allowed flow of tokens between places. Thus, the possible token flows represent the possible evolution of states.

A simple example for a Petri net is shown in Fig. 2.3. In the Figure, $p1$ to $p8$ are places; $t1$ to $t7$ are transitions; arcs are between places and transitions. In the state (also called marking) depicted in the figure, a token is present in place $p1$, and another token is in place $p7$. The next possible state (reachable marking), by executing (firing)

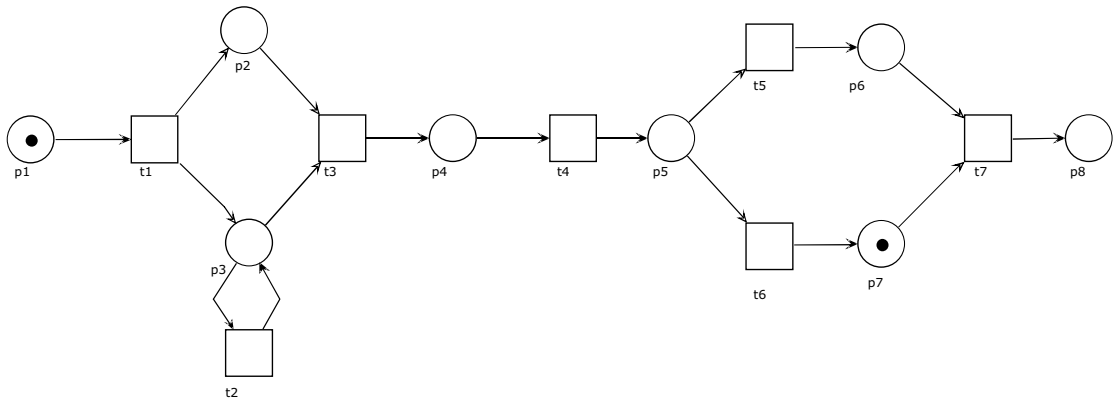

Figure 2.3: Example for a Petri net.

$t1$, would lead to no token present in $p1$, and one token present in $p2$ and one token in $p3$. The token in $p7$ would not be moved anywhere instead.

Petri nets are frequently used to represent the behavior of business processes, since they offer an effective way to model the possible state transitions, which may occur during the execution of a process. An early proposal for the adoption of a variant of Petri nets in workflow systems can be found in [61]. The adoption of Petri nets for modeling business processes is motivated in [138]. In the same work, the author defines Workflow Nets. Workflow Nets are a specialization of Petri nets, which imposes restrictions on the usage of the Petri net formalism to faithfully represent the behavior of process models. An overview on the work supporting the modeling of processes using Petri nets can be found in [129].

Advantages of Petri nets over other formalisms include the support for the simulation, as well as the verification of certain structural and behavioral properties of process models, such as the freedom from deadlocks. In spite of the development of extensions for Petri nets such as the Coloured Petri nets (see [85, 101]) and the Timed Petri nets (see [157]), and their adoption for representing business processes, support for a comprehensive modeling of the various aspects of a process is still lacking within this formalism.

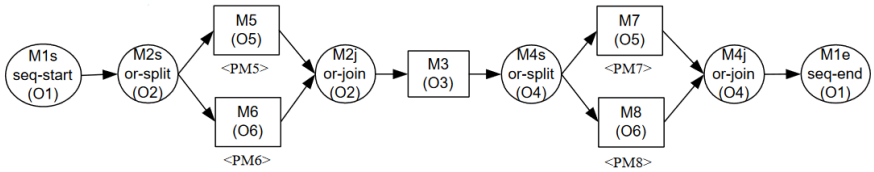

Figure 2.4: Example for a Workflow graph from [53].

Workflow Graphs

Workflow graphs have been introduced in [127, 128] as an alternative to Workflow nets for representing business processes. The goal was to provide a generic language, compliant with the initial indications of the WfMC, which could be used to define and manipulate process models, as well as verifying their structural properties.

Workflow graphs comprise constructs for the representation of control flow elements such as tasks, and or-splits. They also allow modeling both sequential and parallel executions. In the initial work [127], the authors proposed a number of correctness criteria for process models, and verification procedures. These procedures were further refined in [142], with an approach based on transforming Workflow graphs to Workflow Nets.

Workflow graphs as initially introduced mainly capture the functional and behavioral aspects of a process model. However, Workflow graphs have been extended with several works, such as [53, 75], to increase expressiveness and capture, e.g., also the temporal aspect of a process.

For the simplicity and generality of the language, we opt for defining the process model we introduce in Chapter 4 as an extension of a Workflow graph.

BPMN

Business Process Modeling Notation (*BPMN*) [119] was first formalized in 2004, and reached its current version 2.0 in 2011 [1]. It is the *de facto* standard notation for representing process models [150]. The reason behind the success of BPMN lies in having managed to fulfill two main goals at the basis of the development of BPMN: facilitating modeling and communication, and being able to execute process models without any additional operation required. Indeed, one of the reasons for the success

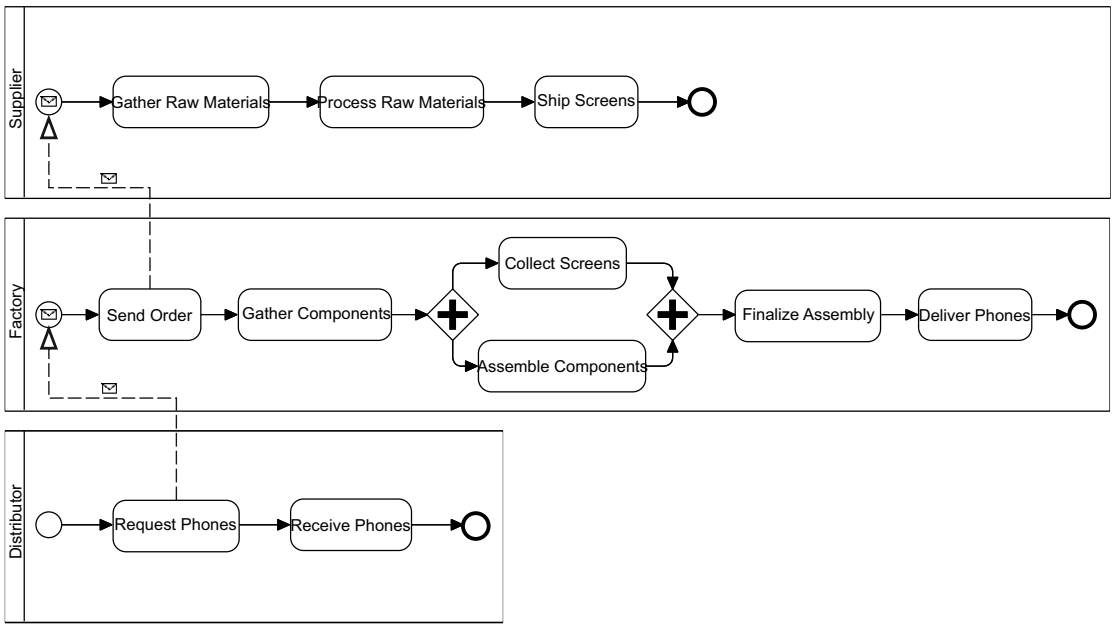

Figure 2.5: Example for a Business Process Diagram for a cross-organizational process.

of BPMN is its intuitiveness, which allows the process models built using BPMN, known as Business Process Diagrams (an example of which is shown in Fig. 2.5), to be understood by stakeholders from the most diverse backgrounds [71]. Another reason is the ease of transformation from a BPMN model into an object which can be directly executed by a workflow engine. A number of commercial tools, such as Camunda [2], are available for the modeling and execution of BPMN-based processes.

BPMN 2.0 offers a large number of elements for modeling business processes. It has been observed, however, that in practice only a limited subset of these elements, with restricted semantics, is actually used in most process models [159]. Nevertheless, despite Business Process Diagrams are able to capture, to different extents, the functional, behavioral, informational, organizational, and operational aspects, numerous extensions to the standard have appeared. The goal of these extensions is primarily of increasing the expressiveness of process models, for instance for representing temporal constraints [21, 69], or for supporting process monitoring [7]. None of these extensions, however, really took off.

[2] `https://camunda.com/`

Research efforts have been devoted to the analysis and verification of BPMN-based process models, e.g., [5, 18, 47, 124]. Most of the proposed approaches for checking the various qualities of process models rely on mapping them into some other formalism. Adequate support to the verification of the temporal correctness of time-constrained BPMN-based process models, however, still lacks.

2.3 Temporal Constraint Networks

The second stream of research at the basis of this thesis comes from research on temporal modeling and temporal reasoning. Modeling and reasoning about time are relevant problems in several application and research areas, such as planning and scheduling, Artificial Intelligence, and Business Process Management. Thus, there is a transversal need for a general language for the modeling of temporal problems, and which supports the attainment of solutions to these problems through mechanisms for temporal reasoning. In this section we introduce the formalism of Temporal Constraint Networks, which, due to its maturity, is our elective language to support temporal modeling and reasoning in business processes.

2.3.1 Temporal Constraint Network Types

Temporal Constraint Networks (TCNs) are a widely adopted formalism for the representation of temporal constraints between points in time, the verification of the temporal correctness of a given set of temporal constraints, and the computation of schedules. At the basis is the idea of adopting a graph-based approach, in which time points are represented as graph nodes, and (binary) constraints between them are represented as weighted graph edges.

Several types of Temporal Constraint Networks, each with different levels of expressiveness, have been proposed over the years as a formal means to represent sets of constraints and perform temporal reasoning on them. Here we give an overview of these types of Temporal Constraint Networks, and discuss their peculiarities.

Simple Temporal Networks

Simple Temporal Networks (STNs) were first introduced in [45] as a formalism to solve the Simple Temporal Problem (STP). In the STP, a set of time points $X_1, ..., X_n$, and a set of constraints in the form of inequalities on the various X_is are given. In particular, given any two time points X_i, X_j, zero or one constraint of the form $a \leq X_i - X_j \leq b$ can be set between them [3]. A solution for the STP is an assignment $X_1 = \overline{X_1}, ..., X_n = \overline{X_n}$ to the time points, such that all the constraints are satisfied. Since each constraint can be split in a pair of inequalities $a \leq X_i - X_j$ and $X_i - X_j \leq b$, solving the STP thus corresponds to solving a set of linear inequalities.

The essential contribution of [45] is the representation of the set of linear inequalities as a directed graph, called *Simple Temporal Network*, in which nodes are used to represent time points, and edges correspond to constraints between time points. In particular, an edge from time point X_i to time point X_j with weight w, represents the constraint $X_j \leq X_i + w$. A Simple Temporal Network is *consistent* if there exists at least one solution to it, i.e. it is possible to find an instantiation for all time points such that all constraints are satisfied. From the observation that such a network forms a *distance graph*, for which the minimum distances between pairs of nodes can be computed, follows the importance of the following theorem, reported in [45], which states the condition for consistency of a STN:

Theorem 2.3.1 (Shostak [133], Liao and Wong [100], Leiserson and Saxe [98]). *A given STP, T, is consistent if and only if its distance graph, G_d, has no negative cycles.*

In essence, the proof of Theorem 2.3.1 is based on the observation that a negative cycle in the network induces a contradiction in the set of inequalities expressed by the edges; and on the fact that the absence of negative cycles means that the shortest paths between pairs of nodes are well-defined, hence the constraints can be satisfied.

As a corollary to Theorem 2.3.1, Dechter et al. show that two consistent solutions are those in which each time point is assigned its latest and earliest possible times d_{max} and d_{min}, respectively. These values are the weights of the edges connecting *zero* to the time point, and the time point to *zero*. Consequently, they also show that the set of feasible values for time points is the interval $[d_{min}, d_{max}]$, and an incremental solution to the STP can be built from a partial solution by selecting appropriate values for time points from these intervals, in such a way that all constraints are satisfied.

[3] This does not exclude the existence of a constraint of the form $a \leq X_j - X_i \leq b$.

Simple Temporal Networks rely on the freedom of choice of an executor for assigning values to all the time points, in order to satisfy all constraints. However, STNs lack the expressiveness to model certain constraints which are guaranteed to hold, but do not allow the executor to choose the values for time points. These constraints are in particular necessary, when uncontrollable durations need to be represented: to this end, a more refined type of temporal network needs to be introduced.

Simple Temporal Networks with Uncertainty

The Simple Temporal Problem under Uncertainty (STPU) has been introduced by Vidal and Fargier in [144, 145], stemming from the consideration that not all durations are under the control of an executor, i.e. the assignments of a time point may not be controlled, but only be observed to fall within a given interval relative to some other time point. These uncontrollable durations are called *contingent*.

As an extension of the STNs, *Simple Temporal Networks with Uncertainty* (STNUs) serve as the formalism proposed in [144, 145] to solve the STPU. A STNU is a graph comprising nodes and edges, which represent either contingent or controllable time points and durations. We give a formal definition for STNU paraphrasing the definitions given by Vidal and Fargier in [144, 145] and by Morris and Muscettola in [115]:

Definition 2.3.1 (Simple Temporal Network with Uncertainty (STNU))**.** *A* STNU *is a tuple* $\langle N, E, l, u, C \rangle$*, where:*

- N *is a set of nodes;*
- E *is a set of edges;*
- $l : E \to \mathbb{R} \cup \{-\infty\}$ *and* $u : E \to \mathbb{R} \cup \{+\infty\}$ *are functions mapping edges into real numbers;*
- C *is a subset of the edges. The edges in* C *are contingent links, and the other edges are non-contingent edges. For each contingent link* e*,* $0 < l(e) < u(e)$*.*

Contingent edges represent intervals whose duration cannot be controlled (using the words of [115], they are *controlled by Nature*), but merely observed: they are, however, subject to the limits imposed by the functions l and u.

With the distinction between non-contingent and contingent edges, STNUs allow capturing the contingent nature of certain durations. With STNUs the notion of consistency becomes less adequate, requiring to define some alternative, more advanced

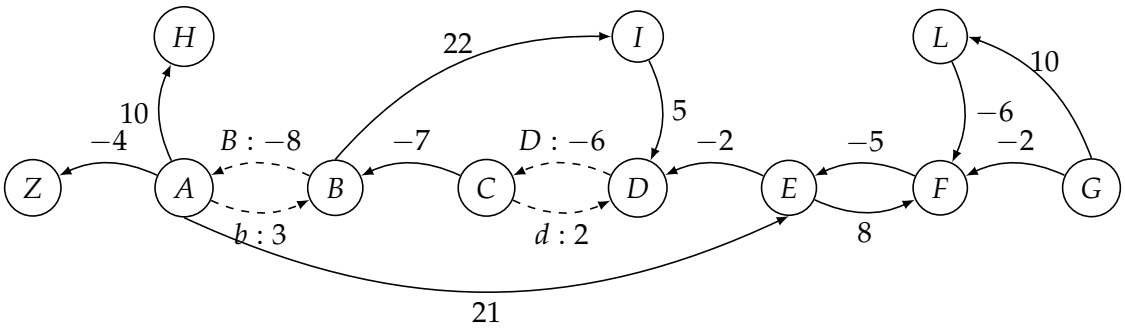

Figure 2.6: Example for a Simple Temporal Network with Uncertainty (STNU).

notion for temporal correctness. However, checking STNU temporal correctness, as will be shown in the next section, still relies on verifying whether they induce negative cycles. Fig. 2.6 shows an example for a STNU: we represent contingent edges as dashed and with a label, following the usual notation for STNUs (see [114]).

Conditional Simple Temporal Networks with Uncertainty

Tsamardinos et al. in [136] raised the objection that there exist application scenarios in which the definition of a time plan depends on the values certain observed conditions may take, determining the presence or absence of certain temporal constraints to fulfil. This observation led to the definition of the Conditional Simple Temporal Problem (CSTP), and the consequent formalization of the *Conditional Simple Temporal Networks* (CSTNs) for modeling the problem.

The two temporal problems, the STPU and the CSTP, were merged by Hunsberger et al. in [82], where a first definition of the *Conditional Simple Temporal Network with Uncertainty* (CSTNU) can be found as a formalism to represent temporal problems which combine observed uncontrollable conditions, and uncertain durations.

Basically, CSTNUs are an extension of the STNUs, which in addition have observation time points, and labels for the time points and edges. Each observation time points is associated to a proposition, which corresponds to an observed condition. Labels are conjunctions of positive or negative such propositions, which in essence are used to represent scenarios. These scenarios are the possible executions of the network, which depend on the conditions observed at the observation time points.

The idea is that each time point and edge is associated with a condition, encoded in the label. This condition reflects the requirement that the corresponding event, resp. constraint, must occur, resp. hold, under the condition expressed in the label. So with this formal apparatus it is possible to model temporal problems, in which certain events and constraints must belong to only certain scenarios.

To better understand the definition of CSTNU given by Hunsberger and Posenato, and for consistency with their notation, we first report here their formalization of the CSTN, on which they build the definition of CSTNU:

Definition 2.3.2 (Conditional Simple Temporal Network (CSTN) [82])**.** *A* CSTN *is a tuple* $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$*, where:*

- $\mathcal{T}$ *is a finite set of real-valued time points;*
- P *is a finite set of propositions (propositional letters);*
- $L : \mathcal{T} \to P^*$ *is a function that assigns a label to each time point in* $\mathcal{T}$*;*
- $\mathcal{OT} \subseteq \mathcal{T}$ *is a finite set of observation time points;*
- $\mathcal{O} : \mathcal{P} \to \mathcal{OT}$ *is a bijection that associates a unique observation time point to each proposition;*
- $\mathcal{C}$ *is a set of labeled simple temporal constraints, each of the form* $(Y - X \leq \delta, \ell)$*, where* $X, Y \in \mathcal{T}, \delta$ *is a real number, and* $\ell \in P^*$*.*

Formally, a CSTNU is defined in [82] as a combination of CSTN and STNU as follows:

Definition 2.3.3 (Conditional Simple Temporal Network with Uncertainty (CSTNU))**.** *A* CSTNU *is a tuple* $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P, \mathcal{L} \rangle$*, where:*

- $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ *is a CSTN, as formalized in Def. 2.3.2;*
- $\langle \mathcal{T}, \lfloor \mathcal{C} \rfloor, L \rangle$ *is a STNU, as formalized in [82], where* $\lfloor \mathcal{C} \rfloor$ *is the set of constraints without the labels, since they do not appear in STNUs;*
- *for each* $(A, x, y, C) \in \mathcal{L}, L(A) = L(C)$*, and* $\mathcal{C}$ *contains labeled constraints,* $(x \leq C - A \leq y, L(A))$*.*

Of course, as demonstrated in [82], since the CSTNU is a formalism built by combining STNUs and CSTNs, any STNU and any CSTN can be represented as a CSTNU, either without any propositional label (in this case, the empty label denoted with the symbol $\boxdot$ is used), resp. without any contingent constraints.

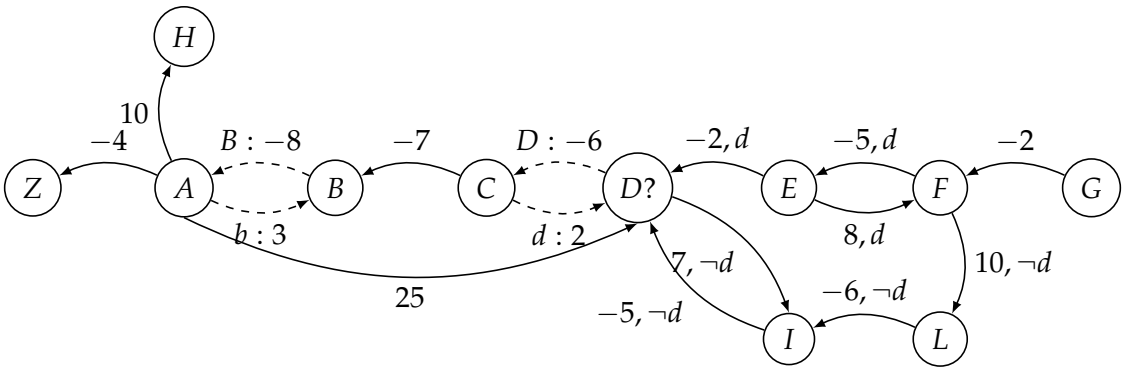

Figure 2.7: Example for a Conditional Simple Temporal Network with Uncertainty (CSTNU).

Fig. 2.7 shows an example for a CSTNU, where $D?$ is an observation time point, in which condition d is observed. Subsequent edges are labeled either with d or $\neg d$, depending on whether they must hold if the observed condition holds, or not. For compact representation, the figure does not show the empty label, which is implicit.

Other Temporal Constraint Networks

To target temporal problems demanding higher expressive power, several other types of Temporal Constraint Networks have been developed. These include the Disjunctive Temporal Network with Uncertainty (DTNU, adding disjunctive constraints) [25], the Conditional Simple Temporal Network with Decisions (CSTND, adding split decision points, in which decision is taken by the executor) [15], the Simple Temporal Network with Uncertainty and Decision (CSTNUD, combining STNU and CSTND) [152], the Simple Temporal Network with Partially Shrinkable Uncertainty (STNPSU, adding flexibility to partially restrict the intervals of contingent durations) [94], or the Simple Temporal Network with Uncertainty and Resources (CSTNUR) [39]. Further analysis of the differences between various types of Temporal Constraint Networks can be found in [11].

Adoption of these more sophisticated Temporal Constraint Networks is not yet as widespread as for the STNUs and CSTNUs. Additionally, the temporal reasoning algorithms designed for these networks do not scale well due to the increased expressiveness, which makes each of these formalisms more complex. Thus, for this work we

limit to considering STNUs and CSTNUs as formalisms for temporal reasoning which provide sufficient expressiveness.

2.3.2 Controllabilities

Due to the existence of contingent durations, the notion of consistency of STNs is not viable in the context of the more expressive temporal networks. Vidal and Fargier therefore defined the Simple Temporal Networks with Uncertainty (STNUs) along with the notions of *strong controllability*, *weak controllability*, and *dynamic controllability* in [144]. In the following we provide definitions for these concepts by paraphrasing the formal definitions given in [144].

Definition 2.3.4 (Strong Controllability)**.** *A STNU N is* Strongly controllable *if and only if there exists an assignment δ of the controllable time points such that, for each possible assignment ω of the contingent time points of N, δ is a solution to N_ω, where N_ω is N in which the contingent time points have been fixed by ω.*

Basically, strong controllability requires the existence of a universal solution, valid for all possible contingent durations, which can thus be computed once and for all. However, such notion may be too restrictive since it give no flexibility of choice for the assignment of controllable time points.

Definition 2.3.5 (Weak Controllability)**.** *A STNU N is* Weakly controllable *if and only if for each possible assignment ω of the contingent time points of N there exists an assignment δ of the controllable time points such that δ is a solution to N_ω, where N_ω is N in which the contingent time points have been fixed by ω.*

The notion of weak controllability, in contrast to strong controllability, requires that the assignment of contingent time points is known, and after observing it, all controllable time points can be assigned a value. For a weakly controllable network, a set of control sequences (assignment of controllable time points) can be computed beforehand for all possible contingent assignments, and applied as soon as their values are known. Such a notion, however, is not applicable to dynamic domains, in which not all contingencies are unveiled at once, but are observed as time goes on.

Definition 2.3.6 (Dynamic Controllability)**.** *A STNU N is* Dynamically controllable *if and only if for each possible assignment ω of the contingent time points of N there exists an assignment δ of the controllable time points such that, for all partial observation of previous*

contingent time points and for all possible future observation of future contingent time points, the current partial assignment of controllable time points can be extended to a solution of N.

The notion of dynamic controllability is suitable for application domains in which an executor, or controller, is required to take action in response to run time observations. In particular, dynamic controllability requires that, for all possible observations of past contingent durations, the executor is able to assign values to the subsequent time points, based on these observations, such that all temporal constraints are met.

How do these three types of controllability relate to each other? In [144] the authors sketch proof of the following Lemma:

Lemma 2.3.1. *Strong controllability* $\implies$ *Dynamic controllability* $\implies$ *Weak controllability.*

The proof is rather straightforward. For the first implication, it is sufficient to observe that if a universal solution exists, then this solution can be dynamically applied. For the second implication, the observation is that a dynamically constructed solution for a given situation, constitutes a full solution for that situation.

Dynamic controllability is regarded as the most adequate notion for temporal correctness in presence of uncertainties. So now comes the question, on how to check whether a given temporal network is dynamically controllable.

2.3.3 Checking Dynamic Controllability

Over the years, several algorithms have been proposed for checking the dynamic controllability of a temporal network. We identify two main approaches: one based on Timed Game Automata, and one based on constraint propagation. Here we give an overview of these approaches.

Timed Game Automata-based Approach

Timed Automata (TAs) have been introduced in [4]. TAs are an extension of finite automata [79] with *clocks* and *clock constraints* to measure time. Timed Game Automata (TGAs) [26] are in turn an extension of TAs, in which the transitions are partitioned into sets of controllable and uncontrollable transitions, to model the interplay between controllable and uncontrollable assignment of timestamps to time points.

The approach to checking dynamic controllability based on Time Game Automata requires mapping a temporal network into a TGA, and checking dynamic controllability on this TGA. The check follows the principle of a two-players game, in which the players' moves correspond to assigning timestamps to time points, realizing transitions. One of the players is the controller (the executor); the other player is the environment (*Nature*). The idea of the dynamic controllability check is then based on verifying the existence of a winning strategy for the controller, to react to the moves (transitions) performed by the opponent. If such a strategy exists, dynamic controllability is ensured. Such an approach to checking dynamic controllability through TGAs is used, for instance, in [26, 152].

Constraint Propagation-based Approach

The most widely adopted approach to verify the dynamic controllability of a Temporal Constraint Network is based on the propagation of the explicit temporal constraints, which are encoded as the network edges. Through constraint propagation, new constraints in the form of implicit edges can be derived, according to a set of constraint propagation rules.

Several sets of constraint propagation rules for deriving implicit edges have been proposed over time, each set targeting a specific type of temporal networks. A set of 5 constraint propagation rules for STNUs has been introduced by Morris and Muscettola in [115, 116]; a more recent set of 3 propagation rules for STNUs constitute the RUL system by Cairo and Rizzi [17]. A significant contribution to the development of CSTNUs comes from the work of Combi, Hunsberger, and Posenato, who propose constraint propagation rules, and analyze the constraint propagation algorithm complexity in [33, 80, 82].

At the basis of the constraint propagation-based approach, is the observation that a given configuration of temporal constraints, which does not lead to a dynamically controllable network, implicitly yields a contradiction. In particular, in a Temporal Constraint Network, a contradiction corresponds to the existence of a negative cycle in the graph resulting from the propagation of constraints [45]. Basically, the approaches which rely on constraint propagation are aimed at verifying whether a negative cycle can be derived from the existing configuration of constraints. If, by iteratively deriving implicit constraints in a Temporal Constraint Network, a negative cycle is found, then the network is not dynamically controllable. Instead, if it is possible to reach network

quiescence (i.e. no new constraints can be derived) with the repeated application of constraint propagation rules, then it is concluded that the network is dynamically controllable.

In Fig. 2.8 and Fig. 2.9 we show the constraint propagation rules defined by Morris and Muscettola, and by Cairo and Rizzi, respectively. The set of rules from Morris and Muscettola comprises 5 rules: *No Case*, *Upper Case*, *Lower Case Cross Case*, and *Label Removal*. These rules, with the exception of *Label Removal*, are applied by considering two consecutive edges and verifying whether some applicability conditions hold for these edges. The rules derive a new edge, which connects the time points at the two extremities of the sequence of edges, if the specified applicability conditions hold. A derived edge expresses a constraint, whose bound depends on the bounds of the original constraints, and may bear an additional label, due to contingencies expressed by the original edges. Labels may be removed from edges, through the application of the *Label Removal* rule.

As an example, the *No Case* rule is applied unconditionally, when two consecutive non-contingent edges connect a time point P to a time point Q, and the same time point Q to a third time point R. The rule derives a new non-contingent edge directly connecting P to R, whose weight is the sum of the weights of the original edges (see Fig. 2.8).

The *Upper Case* rule is applied when a non-contingent edge from P to Q, with weight v, is followed by a contingent edge from Q to A^C, representing the upper-bound of a contingent constraint, with weight w, where A^C is the activation time point of the contingent constraint. If $P \neq A^C$, then the rule can be applied to derive a new contingent edge from P to A^C, with weight $v + w$ (see Fig. 2.8).

The rules of Morris and Muscettola derive new edges, which may be contingent or non-contingent. This has the consequence that potentially a large number of edges may be derived for a given network. The more recent RUL system by Cairo and Rizzi, instead, derives only non-contingent edges. Thus, a comparatively lower number of edges is derived by RUL, since between two time points there can be only one non-contingent edge. Also, a lower number of rules is required for propagating only non-contingent constraints: these are *Relax*, *Upper*, and *Lower*. *Relax* works basically as the *No Case* rule of Morris and Muscettola. The *Upper* rule derives, unconditionally, a non-contingent edge from the same configuration of edges as *Upper Case* of Morris and Muscettola. The *Lower* rule, instead, is applied conditionally, but still derives a

Rule	Graph representation	Applicability conditions
No Case	$P \xrightarrow{v} Q \xrightarrow{w} R$, $P \xrightarrow{v+w} R$	(none)
Upper Case	$P \xrightarrow{v} Q \xrightarrow{C:\,w} A^C$, $P \xrightarrow{C:\,v+w} A^C$	$P \neq C$
Lower Case	$A^C \xrightarrow{c:\,l} C \xrightarrow{w} R$, $A^C \xrightarrow{l+w} R$	$w \leq 0$, $R \neq C$
Cross Case	$A^C \xrightarrow{c:\,l} C \xrightarrow{D:\,w} A^D$, $A^C \xrightarrow{D:\,l+w} A^D$	$w \leq 0$, $C \neq D$
Label Removal	$P \xrightarrow{C:\,z} A^C$, $P \xrightarrow{z} A^C$	$z \geq -l^C$

Figure 2.8: Constraint propagation rules for STNUs developed by Morris and Muscettola. Image from [17].

non-contingent edge, for a configuration of edges like the one considered by *Lower Case* of Morris and Muscettola (see Fig. 2.9).

Rule	Graph representation	Applicability conditions
RELAX	$P \xrightarrow{v} Q \xrightarrow{w} R$, $P \xrightarrow{v+w} R$	(none)
UPPER	$P \xrightarrow{v} C \xrightarrow{C:\,-u^C} A^C$, $P \xrightarrow{\max\{v-u^C,\,-l^C\}} A^C$	(none)
LOWER	$A^C \xrightarrow{c:\,l^C} C \xrightarrow{w} R$, $A^C \xrightarrow{l^C+w} R$	$w \leq 0$ for non-contingent R $w \leq u^R$ for contingent $R \not\equiv C$ with upper-bound u^R

Figure 2.9: Constraint propagation rules for STNUs developed by Cairo and Rizzi. Image from [17].

Chapter 3

Problem Description and Research Questions

In this chapter we discuss process modeling requirements, which cannot be expressed with current languages allowing designers to model temporally constrained processes, and to check their temporal correctness, such as the languages introduced in the previous chapter. We support our discussion with a number of examples. These examples are presented to motivate a lack of expressiveness, which is both highly relevant for applications, as well as formally challenging.

We first present 3 main limitations, with the help of examples. We then present the research questions, which arise from the observation of these limitations, and are at the basis of this work.

3.1 Problem Description

Most of current languages for modeling time-constrained processes suffer from the lack of support for the expression of a large number of temporal constraints. On the other hand, a number of approaches to check temporal correctness, i.e. absence of contradictions or time failures at run time, are available for these languages. However, we advocate for the need for more expressive modeling languages, to provide increased support to practitioners, who need to include real-world temporal requirements into process models.

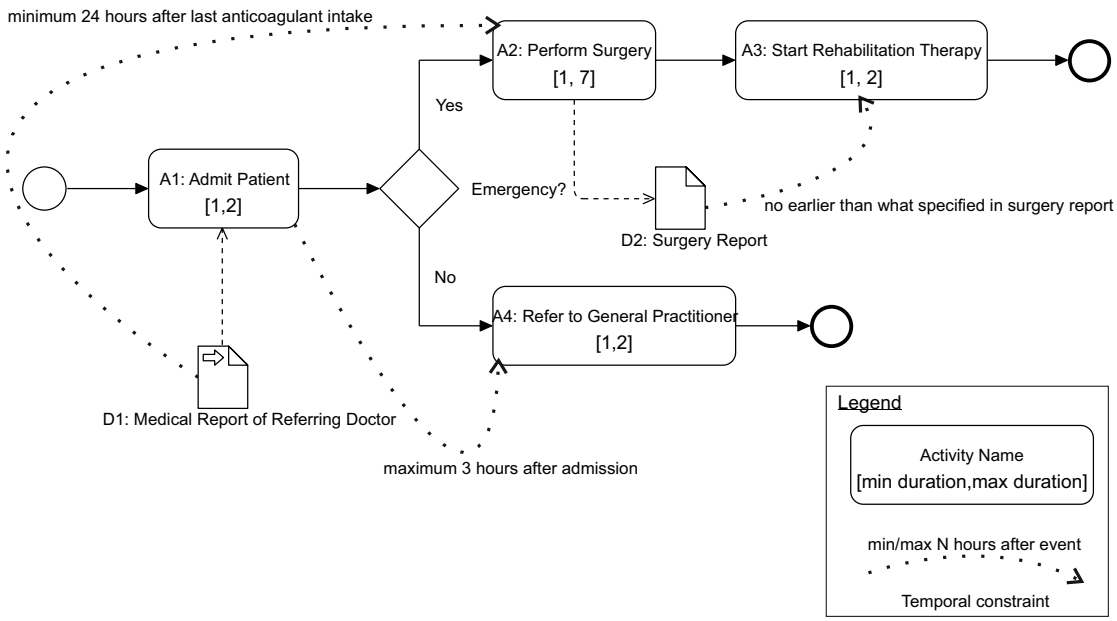

Figure 3.1: BPMN-like fragment of a medical process with temporal constraints.

3.1.1 Dependency on Temporal Data

Most business processes comprehend temporal information and temporal requirements which have to be modeled, resp. fulfilled, for a correct execution. However, there are real-world scenarios for which no formulation of temporal information or constraint is possible with the current modeling approaches. Here we show examples of such cases.

Our first example comes from the medical domain. Suppose a hospitalization process, which starts with the admission of a patient for unplanned treatment. we depict an example for such a process in Fig. 3.1. The process starts with the admission of a patient to the emergency room. A medical report from the referring doctor is provided by the patient. The report includes information about his/her condition, as well as the date of last intake of the anticoagulant drug. If the patient is evaluated not to be in an emergency situation, he/she is sent back to the general practitioner within 3 hours after admission (deadline fixed for all non-emergency patients by hospital policies). If it is an emergency, the patient needs to undergo a surgery. During the surgery a report is filled in by the surgeon, in which the latest time for post-surgical recovery therapy is indicated. Finally, the therapy starts.

The following temporal constraint has to be fulfilled: *Emergency surgery can only be performed at least 24 hours after the patient had the last intake of an anticoagulant drug*. With current formalisms, this constraint can only be expressed if the event *intake of a drug* is part of the process. However, in this example *intake* is not an activity of the treatment process, since it might have happened before the process started. Therefore, we cannot express this constraint with current modeling languages.

How are such situations handled in real life? Well, we capture the relevant temporal information in a data item and use this data item to formulate temporal constraints and to pass the required information with this data item to the process execution. In our example, the time of the last intake could be obtained from an interview with the patient when she is admitted to the hospital, and this time point is registered in the admittance form. Alternatively, we could take the time of the last intake from the clinical record of the patient. In any case, we then must use stored data as a temporal input for scheduling the treatment process.

Here are some more examples of temporal constraints which cannot be expressed by formalisms, which only allow the specification of constraints between control flow events:

- The delivery of an ordered item has to take place in the time interval indicated by the customer in the order form.
- Recovery therapy can start no earlier than the date specified by the surgeon in the surgery report.
- A one-time password can be used for logging in into a system no later than the expiration date stamped in a message received with the password.

The general pattern of all these examples is that it would be necessary to express some temporal constraint as a temporal relationship between a data item and a control flow event. We are not considering the references to data items in conditions determining the control flow, but concentrate on the use of data items in temporal constraints. The specialty of these data items used in temporal constraints is that they may influence in particular the scheduling of a process, rather than the control flow. For instance, in the example of Fig. 3.1, the actual date of the last anticoagulant intake does not influence the decision about the emergency, but it may require that the surgery is postponed, in order to fulfill the temporal constraint.

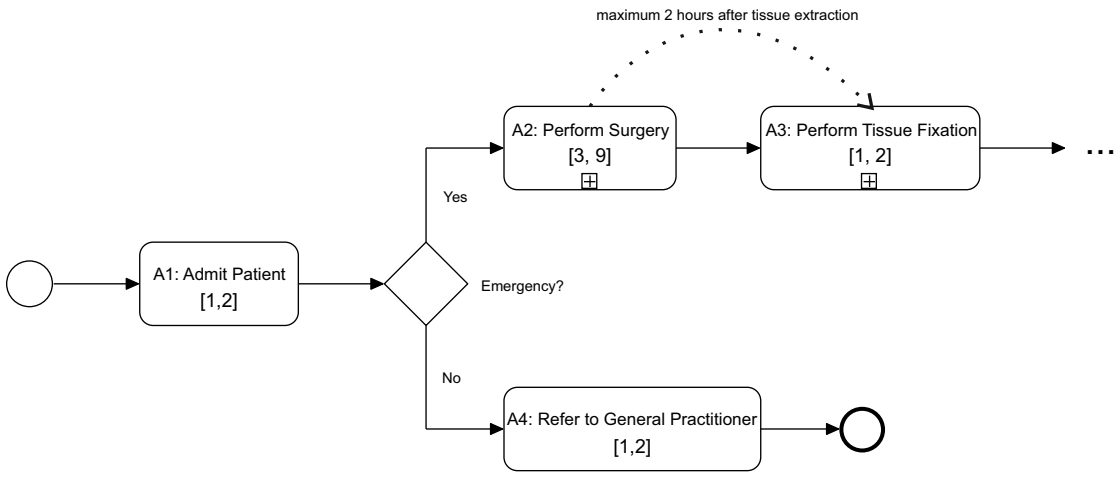

Figure 3.2: BPMN-like fragment of a modularized process with temporal constraints.

3.1.2 Temporal Constraints Across Modules

A problem around the formulation of temporal constraints arises also with modular processes. For a simple example, consider the simplified modularized process in Fig. 3.2. In the process, during surgery a biopsy is taken, the tissue transferred to the pathology department where the biopsy is fixated. Both the surgery and the treatment of the biopsy in the pathology department are modules (subprocesses). Quality management requires that the sample has to be fixated no more than 2 hours after taking the biopsy. With current modeling languages, we either cannot express this requirement in the process model, or we have to break up the modules to exhibit the relevant events.

The advantages of modularizing processes are obvious and well studied [125, 126, 158]. These advantages include improved structuring of processes, subprocesses are reusable in different other processes, process models are more comprehensible and manageable, etc. With the existing approaches, however, temporal constraints have to be bound to an executable and visible process event. Since events are only visible if they are within the same module, it is not possible to specify any temporal constraint crossing module boundaries. This either reduces the possibilities for modularization, or reduces the possibility of formulating, and therefore considering or enforcing, temporal constraints. Again the solution can be that temporal information is captured in

form of data. This data can be passed between modules as parameters. Then these parameters can be used to formulate and enforce temporal constraints.

3.1.3 Temporal Constraints Across Distributed Processes

A similar problem can be observed in the modeling of distributed, or cross-organizational, processes [12, 22, 56] realized by the interaction of local processes executed by different process partners, in which there is a need to express temporal constraints across these local processes. An example for such a need is shown in Fig. 3.3.

In the example, three organizations execute their local processes: a Distributor of mobile phones, a Factory producing mobile phones, and a Supplier of raw materials for building mobile phone screens. Each organization may have its own (temporal) requirements for the execution of their local process. However, an organization may also have additional requirements for the collaborating parties. For instance, the Distributor may require the Factory to complete the delivery of the mobile phones at most 21 days after the order has been placed; similarly, the Factory may require the Supplier to ship the screens at least 8 days after it starts to gather other components.

Such temporal constraints across the local processes can only be expressed, if more internals of those processes are exposed than otherwise necessary. This is usually not desired, since it means that flexibility is reduced and coupling is made stronger. Furthermore, privacy is reduced and there is the risk of a potential exposition of business secrets. Like in the case of modules, currently there is no way to directly refer to temporal information originating outside of a local process. However, if we can capture the relevant temporal information in form of data, we can model the collaboration such that this temporal data is passed between the interacting local processes, and we can formulate the temporal constraints locally by referring to the received or sent temporal data items.

3.2 Research Questions

Having described the problem with the support of examples, we can now discuss the arising Research Questions, which we want to address with this work. We organize them as 3 main Research Questions, with one question concerning the modeling of

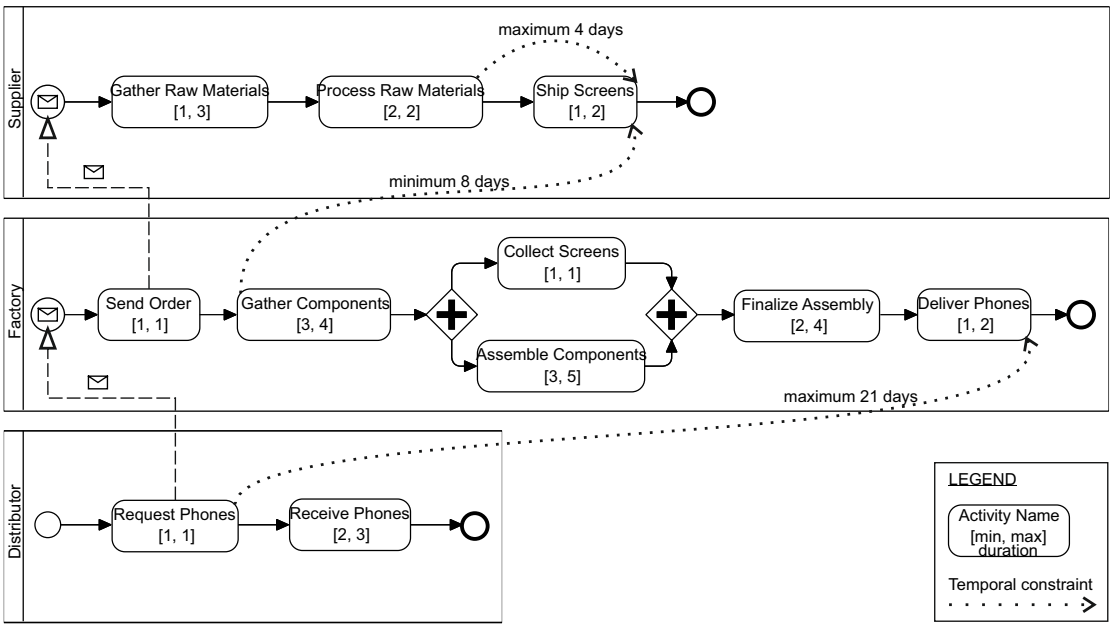

Figure 3.3: BPMN-like fragment of a cross-organizational process with temporal constraints.

processes, one concerning the checking for temporal correctness of processes, and one concerning the computation of additional temporal knowledge.

For each Research Question we identify a number of Research Sub-questions, as follows:

- **RQ1:** How can we support the modeling of the temporal aspects of business processes with data elements such as temporal variables?

  Chapter 4 is devoted to answering this first Research Question. In particular, it addresses the following Research Sub-questions:

  - **RSQ1.1:** What are the elements constituting a process model which features temporal variables to model events as data elements?

    The first contribution of Chapter 4 is the introduction of a fist process model for intra-organizational processes featuring temporal variables for representing temporal data and imposing temporal constraints on them. We then define extensions for this process model, which support the representation of modularized processes, Web service compositions, and local processes

participating in cross-organizational processes. Thus, as an answer to the first Research Sub-question, we produce three artifacts in terms of formal definitions of process models.

- **RSQ1.2:** Which is the enabled expressive power of a process model which features temporal variables to model events as data elements?

  To support the applicability of our process models, and answer the Second research Sub-question, we show in Chapter 4 examples for patterns of temporal constraints, which are enabled by the introduction of temporal variables in the modeling and enactment of process models, and are not expressible with existing modeling languages.

- **RSQ1.3:** Which are the temporal semantics of a process model which features temporal variables to model events as data elements?

  In Chapter 4 we define the concept of *scenario*, to define the temporal semantics of the introduced process models. We additionally define when a scenario is valid, to establish the admissible instances of the process models. Thus, the formal definitions of *scenario* and *valid scenario* are the artifacts which we provide, to answer the third Research Sub-question.

- **RQ2:** How can we check the temporal correctness for process models which feature temporal variables?

  In Chapter 5 we first motivate dynamic controllability as our chosen notion for temporal correctness of process models with temporal variables. We then motivate the choice of Temporal Constraint Networks as the underlying formalism to enable the check dynamic controllability. We show that we can check dynamic controllability of process models by mapping them into temporally equivalent Temporal Constraint Networks. In particular, we answer the following Research Sub-questions:

  - **RSQ2.1:** Which is the most adequate notion for temporal correctness of process models with temporal variables?

    In Chapter 5 we motivate dynamic controllability as the most adequate notion for temporal correctness of process models with temporal variables.

  - **RSQ2.2:** How to check the dynamic controllability of a process with temporal variables?

We show in Chapter 5 how it is possible to map a process into a temporally equivalent Temporal Constraint Network, and adopt dynamic controllability checking techniques for temporal networks, in order to check the dynamic controllability of a process. As an artifact to answer the Research Sub-question, we give a definition, in natural language form, of a procedure, based on mapping processes into Temporal Constraint Networks, for checking the dynamic controllability of processes built after the process model definitions introduced in Chapter 4. We then provide, as an artifact, the implementation of this procedure, which can be used in conjunction with existing tools for verifying dynamic controllability of Temporal Constraint Networks.

- **RSQ2.3:** Which Temporal Constraint Network types are the most suitable, for checking the dynamic controllability of processes which feature temporal variables?

  In the chapter we motivate STNUs and CSTNUs as the currently most adequate temporal networks to use as a formal framework for the dynamic controllability check, discussing pros and cons for each of the 2 types.

- **RSQ2.4:** How to map a process with temporal variables into a Temporal Constraint Network?

  In Chapter 5 we define mapping rules which have the effect of projecting the temporal aspect of a process model with temporal variables into elements of a STNU or CSTNU, realizing a temporal network with the same temporal semantics as the original process model. The artifacts we provide as an answer to the Research Sub-question are formal definitions of two sets of mapping rules, which transform processes, built after the process model definitions introduced in Chapter 4, into two types of temporally equivalent Temporal Constraint Networks.

- **RQ3:** How can we support process designers in the design of temporally correct processes which feature temporal variables?

  The support to the design of processes featuring temporal variables needs to be such that temporal correctness, as defined in Chapter 5, is guaranteed. In particular, this requires helping designers in establishing restrictions for the values temporal variables may take, which offer such guarantees of temporal correctness. In Chapter 6 and Chapter 7 we consider the particular cases of Web service

compositions and cross-organizational processes as motivating scenarios, and address the following Research Sub-questions:

- **RSQ3.1:** How to compute, which are the allowed restrictions for the temporal variables used as input parameters of a process, such that when these variables assume values compatible with these restrictions, dynamic controllability is ensured?

  We motivate in Chapter 6 the use of Temporal Constraint Networks, and in particular constraint propagation techniques, to address the need to compute restrictions for temporal variables holding dynamic controllability guarantees. The artifacts we provide to answer the Research Sub-question is the definition in natural language, and the implementation, of two procedures to compute restrictions for temporal parameters, which are based on the propagation of constraints in Temporal Constraint Networks.

- **RSQ3.2:** Which constraint propagation techniques need to be applied, for computing restrictions for temporal variables, which ensure dynamic controllability of a process using these variables?

  In Chapter 6 we propose a technique, which is built on a recent work aimed at computing allowed values for the contingent links of a Temporal Constraint Network. We show that such a technique is sound and complete for computing allowed restrictions for a single temporal variable of a process model, or for multiple variables which are independent from each other.

  In Chapter 6 we also introduce new constraint propagation rules for Temporal Constraint Networks, which allow us to derive restrictions for temporal variables used as input parameters of a process, which are independent of contingencies in the process definition. We introduce a sound and complete procedure, which uses these rules in order to derive temporal variables restrictions. We show that this procedure may be used also in the case of multiple parameters, with potential dependencies between each other.

  We compare the two proposed approaches, and motivate the second approach as the most suitable.

  So the artifacts we introduce to answer RSQ3.2 are: a function to compute labels of edges in Temporal Constraint Networks; a procedure, based on propagating constraints whose label is computed through our proposed

function, for deriving admissible range of values for temporal parameters; a set of new constraint propagation rules for STNUs; and a procedure, based on the iterated application of these rules, for computing temporal parameter restrictions.

- **RSQ3.3:** How to support the negotiation of computed restrictions on temporal variables, when these temporal variables are shared among several communicating processes, each having different requirements on a same variable?

  In Chapter 7 we build on top of the second restriction computation procedure proposed Chapter 6, and present a distributed procedure to support the negotiation of computed restrictions on temporal variables. We propose the distributed procedure for the negotiation of temporal variables restrictions as the artifact to answer RSQ3.3.

We begin with tackling the first research question around the support to modeling business processes in the next chapter.

Chapter 4

Process Models with Temporal Variables

4.1 Introduction

We propose to extend the modeling of processes with temporal data objects, and to allow the formulation of temporal constraints involving both events and temporal variables. With this we can overcome many of the limitations we described in Chapter 3.

In this chapter we define the process architecture, execution context, and assumptions. We then formalize process models, which feature temporal variables for encoding time information. These process models enable the modeling of intra-organizational processes, modularized processes, Web service compositions, and local processes in cross-organizational processes. We define the temporal semantics of the process models, and show the new patterns of temporal constraints which these process models allow.

The contents of this chapter are based on the publications in [51], [66], [67], [68].

4.2 Architecture, Context, and Assumptions

In this work we target block-structured acyclic processes. With this restriction we are allowed to reason about the temporal qualities of processes, without having to address the consequence of potential design flaws, as pointed out in [28]. Additionally, for broadest applicability, we consider the most frequently used control flow constructs shown in [159]. These control flow constructs include activities, parallel splits and

joins, and exclusive splits and joins. Without loss of generality, we consider exclusive splits which model binary conditional executions, i.e. based on the observation of truth values which can be either `True` or `False`.

Time in a process is measured in an atomic time unit, which we call *chronon*, like minutes, hours, or days, and is represented through natural numbers. Events are stamped with time-points, i.e. points on an increasing time axis representing the temporal distance to a given reference point *zero*.

Processes have specified minimum and maximum task durations, deadlines, and may include additional temporal constraints, stating restrictions between events and time-points.

4.2.1 Task Durations

We define a task duration as the distance on the time axis between the registered time-points for the start and end events of the task. For task durations we distinguish between contingent and non-contingent. In the following we will use the designation contingent, resp. non-contingent task, as a shorthand for task with contingent, resp. non-contingent duration.

A contingent task is a process task whose duration can only be observed, at run time, to take between a guaranteed minimum and maximum duration, but neither the executor, nor the execution environment, can control it. Examples for contingent tasks are the following:

- Bank money transfers: in the European Union, the time for a money transfer requires up to 4 working days for the bank system to complete the transfer. Typically, one day is the least necessary time to wait for the operation to complete. However, a client has no control over the duration of the transfer, and can merely observe when it completes.
- Chemical reactions: in several settings, the time required for a chemical reaction to terminate depends on factors, such as temperature or presence of other compounds, which are out of the control of an observer or experiment conductor, thus making the completion time not controllable.
- Assistance interventions: Service Level Agreements may stipulate maximum response times, e.g., 24 hours, in the case of on-site assistance requests. However,

the actual time of intervention may depend on factors such as distance, or pre-existing workload, which make the response time not controllable.

A non-contingent task, on the other hand, is a process task whose duration can be controlled by the executor, such that the task is completed in an actual time between the specified minimum and maximum durations. Examples for non-contingent tasks are:

- Writing an email: while a minimum duration is sufficient for writing the essentials, more time can be spent improving the quality of the text.
- Giving a talk: a speaker can decide how long to talk about a specific topic within the time interval of 15 to 20 minutes allowed for presentation.
- Testing a student: an examiner may decide how long to keep interrogating a student.

If a task is contingent or non-contingent depends on who is in control of its execution. Thus, a contingent task cannot be made non-contingent, i.e. controllable, for an observer of the task duration. For instance, it is not possible, for a client, to enforce the completion date for a money transfer, since the transfer is done by the bank. However, in [50] it is shown that, from a process model perspective, non-contingent process tasks can be transformed into contingent ones. This transformation is without loss of information or change in the process semantics, and it is realized with the introduction of additional temporal constraints. Consequently, without loss of generality, here we consider processes with only contingent task durations. This assumption allows for a simpler process model, in which we do not need to distinguish between the two types of durations.

4.2.2 Temporal Constraints

Temporal constraints are used to establish restrictions on the distance on the time axis between time-points. We consider binary constraints, i.e. constraints between pairs of time-points. These constraints are either upper-bound or lower-bound constraints. An upper-bound constraint establishes a maximum distance between two time-points on the time axis; a lower-bound constraint establishes a minimum distance. Examples for an upper- and for a lower-bound constraint are the following, respectively:

- In a manufacturing process, it may be required that a polishing task gets executed no more than 5 minutes after a molding task has been completed. Such a requirement can be modeled by establishing an upper-bound constraint from the end event of the molding task to the start event of the polishing task, requiring a maximum distance of 5 minutes on the time axis.
- In the same manufacturing process, before the molding task can start, at least 80 minutes have to elapse after the completion of a previous grinding task. This requirement can be modeled with a lower-bound constraint between the end event of the grinding task and the start event of the molding task, imposing a minimum time distance of 80 minutes on the time axis.

It is easy to see that a process deadline can be encoded as an upper-bound constraint between the start and the end events of the process.

4.2.3 Temporal Variables

To enable the formulation of temporal constraints referring to time-points which do not correspond to task events, we extend the process model with temporal variables.

Temporal variables are data objects of type *timestamp*, which encode temporal information about events or other time-points. Temporal variables can be written and read by process tasks. Each process task implicitly writes two temporal variables, one encoding the timestamp of the start and one encoding the timestamp of the end event of the task. Other temporal variables can be written as well, such as the minimum timestamp for the start event of a subsequent task. Temporal variables can also be received from the environment as temporal parameters. So temporal parameters encode events which are external to the process, or other time-points, such as the expiration date of a password, which can be communicated to a process.

Additionally, temporal variables can be used by a process modeler to formulate temporal constraints, and by the process enactment system to enforce these constraints. Thus, it is possible to establish temporal constraints between task events, and events which are not bound to the occurrence of a task event. Examples for constraints involving variables are the following:

- A one-time password is received from an external service, and may be used for a login task no later than the expiration date which was attached to it.

- During a surgery task, the surgeon can state, through a temporal variable *earliest_physio*, the earliest date a physical rehabilitation may start. *earliest_physio* can then be used in a lower-bound constraint, which binds it with the start of the rehabilitation task.

In Sect. 4.6 we present, with the help of examples, the constraint patterns, which our process model enables due to the extension with temporal variables.

Temporal variables may be restricted in their admissible values through range constraints, which specify minimum and maximum values. Such range constraints may be used by a process designer to force the temporal variables to take only certain values, e.g., which guarantee absence of time failures in the process, as it is shown in [51] and will be discussed in detail in Chapter 6 and Chapter 7.

To avoid concurrency management due to parallel writing of temporal variables, and handling variables with different values over time, here we assume, without loss of generality, that for each temporal variable there is exactly one writing task, and there may be multiple reading tasks.

We now introduce a process model, featuring temporal variables and temporal constraints, for representing intra-organizational processes, and further extend it to support the modeling of modularized processes, Web service compositions, and cross-organizational processes.

4.3 Intra-Organizational Processes

4.3.1 An Example for an Intra-Organizational Process

We take again an example from the medical domain similar to the one depicted in Fig. 3.1, which we report here in more refined form in Fig. 4.1. The process starts with the admission of a patient for unplanned treatment. It is required that the following temporal constraint is fulfilled: *Emergency surgery can only be performed at least 24 hours after the patient had the last intake of an anticoagulant drug*.

We notice that current formalisms can only be express this constraint if the event *intake of anticoagulant drug* is part of the process, and in this example *intake of anticoagulant drug* is not an activity of the process, since it might have happened before the process started. Another example is the second temporal constraint of the treatment

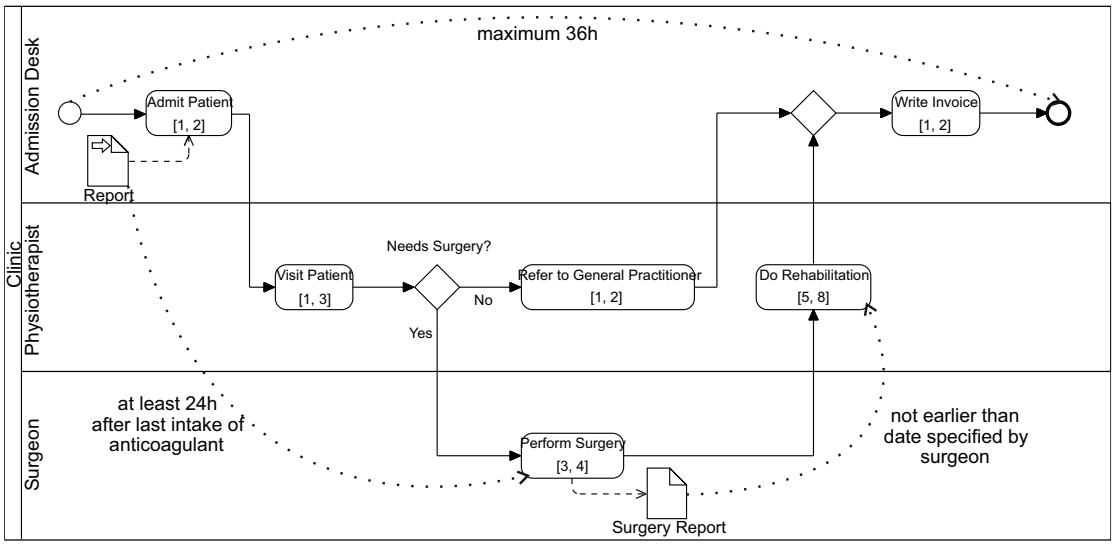

Figure 4.1: Medical process with time information treated as data.

process. The constraint requires that, in case of surgery, the subsequent rehabilitation activity must begin not earlier than the date, which is indicated by the surgeon.

So current modeling languages fall short for modeling these constraints [96]. In real life, the relevant temporal information is stored in a data item (the report from the referring doctor), and this data item is used to formulate temporal constraints, and to pass the required information to the process execution.

In the example, the time of the last anticoagulant intake could be obtained from an interview with the patient during admittance, registering this time point in a form, or the time of the last anticoagulant intake may be received from the general practitioner. Similarly, the earliest time for rehabilitation therapy is stated by the surgeon in a surgery report. In such cases, what we need is to use stored temporal data for verifying whether a schedule is possible, and for computing such schedule.

In the following we introduce a process model featuring temporal variables, which enables modeling constraints such as the ones shown above.

4.3.2 Intra-Organizational Process Model with Temporal Variables

Here we formalize a timed process model with temporal variables in form of an extended graph, which allows the expression of temporal constraints between process events and other time-points under the assumptions presented in Sect. 4.2.

Definition 4.3.1 (Process Model with Temporal Variables). *A process model with temporal variables P is a tuple $(proc_id, N, E, V, C, \Omega)$, where:*

- *$proc_id$ is a unique process id.*
- *N is the set of nodes. A node n has type $n.type \in \{start, activity, xor-split, par-split, xor-join, par-join, end\}$. A node n is associated with two events: $n.s$ for the start, and $n.e$ for the end of the node execution.*
  - *A node n with $n.type = activity$ has a read set $n.r$ of temporal variables read by it, and a write set $n.w$ of temporal variables written by it.*
  - *A node n with $n.type = xor-split/xor-join$ has an associated xor-condition $n.cond$.*
- *$E \subseteq N \times N$ is the set of edges defining precedence constraints.*
  - *An edge e has an associated condition $e.cond$.*
- *V is the set of temporal variables, partitioned in V^T, N^e:*
  - *V^T is the set of temporal variables written by some activity node: $V^T = \bigcup_{n \in N: n.type = activity} \{n.w\}$;*
  - *N^e is the set of temporal variables encoding start and end events of nodes: $N^e = \bigcup_{n \in N} \{n.s, n.e\}$.*
- *C is the set of temporal constraints consisting of*
  - *duration constraints for each $n \in N : d(n, d_{min}, d_{max})$, imposing for the actual duration $\overline{d}$ of n that $d_{min} \leq \overline{d} \leq d_{max}$;*
  - *range constraint for temporal variables $v \in V^T : r(v, v_{min}, v_{max})$, imposing for the actual value $\overline{v}$ of v that $v_{min} \leq \overline{v} \leq v_{max}$;*
  - *upper-bound constraints: $ubc(a, b, \delta)$, where $a, b \in V, \delta \in \mathbb{N}$, imposing for the actual values $\overline{a}, \overline{b}$ of a, b that $\overline{b} \leq \overline{a} + \delta$;*

  - *lower-bound constraints:* $lbc(a, b, \delta)$, *where* $a, b \in V, \delta \in \mathbb{N}$, *imposing for the actual values* $\overline{a}, \overline{b}$ *of* a, b *that* $\overline{b} \geq \overline{a} + \delta$.

- Ω *is the process deadline, imposing that the process ends no more than* Ω *time units after start.*

With the write set we consider an activity n to be the writer of all temporal variables in $n.w$, without making any assumption on its semantics. In particular, we do not distinguish whether a variable $v \in n.w$ is actually written by n or it is received by n as an input parameter.

Each node n with type $xor - split$ (and its corresponding join) is associated with an observed split condition $n.cond$. For a $xor - split$ node n having an edge $(m_0, n) \in E$ with $(m_0, n).cond = c$, the outgoing edges (n, m_1), (n, m_2) are labeled with $c \wedge n.cond$ and $c \wedge \neg n.cond$ respectively. For a $xor - join$ node n associated with a condition $n.cond$, having edges (m_1, n), $(m_2, n) \in E$ with conditions $c \wedge n.cond$ and $c \wedge \neg n.cond$ respectively, the outgoing edge (n, m_3) has condition $(n, m_3).cond = c$. Finally, each edge $(start, n)$ has condition $(start, n).cond = True$.

With duration, upper-, and lower-bound constraints, possibly in combination, and temporal variables, we enable the formulation of a number of time patterns identified by Lanz et al. in [96]. In particular, we enable the expression of patterns identified in [96] by the names *TP1 - Time lags between activities*, *TP2 - Durations*, *TP3 - Time lags between events*, *TP4 - Fixed date elements*, *TP7 - Validity period*.

With the following definition, we set a requirement on a process model to be considered fully specified:

Definition 4.3.2 (Fully Specified Process). *Let* $P = (proc_id, N, E, V, C, \Omega)$ *be a process model specification defined as per Def. 4.3.1.* P *is* fully specified *if,* $\forall v \in V^T, \exists r(v, v_{min}, v_{max}) \in C$.

How can the process from the example in Fig. 4.1 be represented with the process model of Def. 4.3.1? Its elements may be represented as follows:

- $proc_id = example_medical$;

- $N = \{start, Admit_Patient, Visit_Patient, Split, Refer_to_GP, Perform_Surgery, Do_Rehabilitation, Join, Write_Invoice, end\}$;

- $E = \{(start, Admit_Patient), (Admit_Patient, Visit_Patient), (Visit_Patient, Split), (Split, Refer_to_GP), (Split, Perform_Surgery), (Perform_Surgery, Do_Rehabili -$

$tation), (Refer_to_GP, Join), (Do_Rehabilitation, Join), (Join, Write_Invoice), (Write_Invoice, end)\}$;

- $V = V^T \cup N^e$, where:
  - $V^T = \{last_intake_time, earliest_therapy_start\}$;
  - $N^e = \{start.s, start.e, Admit_Patient.s, Admit_Patient.e, \ldots, Write_Invoice.s, Write_Invoice.e, end.s, end.e\}$;
- $C = \{d(start, 0, 0), d(Admit_Patient, 1, 2), d(Visit_Patient, 1, 3), d(Split, 0, 0), d(Refer_to_GP, 1, 2), d(Perform_Surgery, 3, 4), d(Do_Rehabilitation, 5, 8), d(Join, 0, 0), d(Write_Invoice, 1, 2), d(end, 0, 0), r(last_intake_time, 0, 24), r(earliest_therapy_start, 0, 48), lbc(last_intake_time, Perform_Surgery.s, 24), lbc(earliest_therapy_start, Do_Rehabilitation.s, 0)\}$;
- $\Omega = 36$.

The model of Def. 4.3.1 is capable of capturing the relevant aspects of an intra-organizational process model. However, the model can easily be extended, to allow the representation of modularized processes and Web service compositions, as well as local processes which are part of a cross-organizational business process.

4.3.3 Modularized Processes

A modularized process is a process including one or more sub-processes which are treated as modules. A module may feature temporal constraints, which refer to events outside the module itself: these events may occur in the outer process (i.e. the process which contains the module), or in other modules part of the outer process. Opening up the modules, exposing their internals, to express such temporal constraints would be a violation of the very idea of modularization, which is having a black-box paradigm to keep lean models and hide implementation details. However, the introduction of the model with temporal variables of Def. 4.3.1 allows us to overcome this dilemma.

For modularized processes, a process node of type *activity* from Def. 4.3.1 can be seen as a complex activity, encapsulating a sub-process as a module. This sub-process is in turn a process model defined after Def. 4.3.1. Lower- and upper-bound constraints between start and end of the sub-process determine the minimum and maximum durations for the activity encapsulating it. Activities in the process model have a read and a write set, containing temporal variables which are read, resp. written, by

these activities. Thus, variables in these sets may be used as input (read set), resp. output (write set) temporal parameters exchanged between modules in a modularized process. Inside a module, input temporal parameters are used as temporal variables, which are instantiated at module instantiation time; output temporal parameters as temporal variables, which are assigned a value by some local activity.

Temporal constraints referring to temporal variables can be set locally in a module, to realize temporal constraints referring to events outside the scope of the module. Thus, by exchanging temporal variables as parameters between activities encapsulating modules, it is possible to realize temporal constraints in modularized processes without harming module integrity.

4.4 Web Service Compositions

Web service compositions are a widely adopted technology to realize business processes [99]. They create value through the structured combination of Web services.

Temporal Service Level Agreements (T-SLAs) are contracts, in which service providers agree to provide some defined service quality, such as a timely delivery. T-SLAs for Web service compositions can be elegantly defined by temporal parameters encoding temporal information and expressing temporal requirements, restrictions, limitations, and objectives as constraints on temporal parameters.

Similar to the approach to modeling modularized processes, a service provider may model a Web service composition as a process definition after Def. 4.3.1. In this process definition, each process node of type *activity* represents a service call. The duration constraint for such a node represents the (uncontrollable) time span required for the service call to return.

Temporal parameters may be exchanged between Web services in a composition to establish temporal constraints across service calls, thus realizing the T-SLAs. Like for modules, parameter exchange may be implicitly modeled through the read and write sets of activities. Temporal constraints across Web services can thus be realized through local constraints which refer to exchanged temporal parameters.

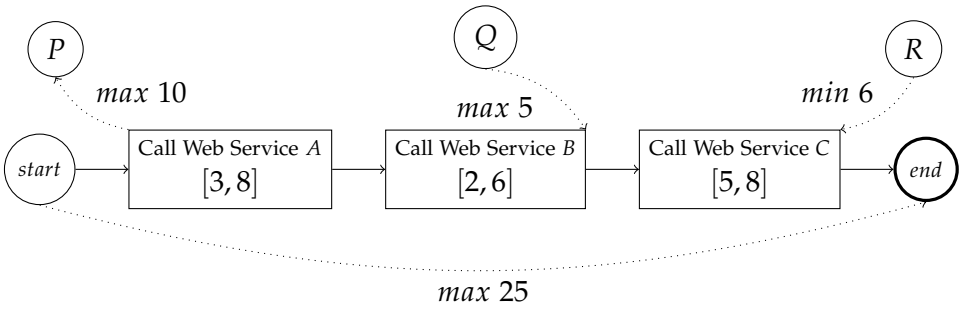

Figure 4.2: Example Web service composition with temporal parameters.

4.4.1 An Example for a Web Service Composition

Consider the simple Web service composition shown in Fig. 4.2, which invokes a sequence of 3 other Web services. First, a service A, which is guaranteed to take between 3 and 8 time units, is called. This first service call must be done no earlier than 10 time units before the timestamp specified in parameter P. After A completes, a service B is called, which requires between 2 and 6 times units. B must complete at most 5 time units after the timestamp specified in parameter Q. Finally, a service C is called, taking between 5 and 8 time units. C must be completed such that at least 6 time units are left between its time of completion and the timestamp specified in parameter R. An additional constraint states that the overall execution must take no more than 25 time units.

4.4.2 Process Model for Web Service Compositions with Temporal Parameters

We define a process model for representing Web service compositions which is similar to the process model for intra-organizational processes of Def. 4.3.1.

Definition 4.4.1 (Process Model for Web Service Composition with Temporal Parameters)**.** *A process model for a Web service composition with temporal parameters P is a tuple $(proc_id, N, E, V, C, \Omega)$, where:*

- *$proc_id$ is a unique process id.*
- *N is the set of nodes. A node n has type $n.type \in \{start, call, xor-split, par-split, xor-join, par-join, end\}$. A node n is associated with two events: $n.s$ for the start, and $n.e$ for the end of the node execution.*

  - *A node n with $n.type = call$ has a read set $n.r$ of temporal variables read by it, and a write set $n.w$ of temporal variables written by it.*
  - *A node n with $n.type = xor - split/xor - join$ has an associated xor-condition $n.cond$.*

- *$E \subseteq N \times N$ is the set of edges defining precedence constraints.*
  - *An edge e has an associated condition $e.cond$.*

- *V is the set of temporal variables, partitioned in V^T, N^e:*
  - *V^T is the set of temporal variables output of some service call node: $V^T = \bigcup_{n \in N: n.type = call} \{n.w\}$;*
  - *N^e is the set of temporal variables encoding start and end events of nodes: $N^e = \bigcup_{n \in N} \{n.s, n.e\}$.*

- *C is the set of temporal constraints consisting of*
  - *duration constraints for each $n \in N : d(n, d_{min}, d_{max})$, imposing for the actual duration $\overline{d}$ of n that $d_{min} \leq \overline{d} \leq d_{max}$;*
  - *range constraint for temporal variables $v \in V^T : r(v, v_{min}, v_{max})$, imposing for the actual value $\overline{v}$ of v that $v_{min} \leq \overline{v} \leq v_{max}$;*
  - *upper-bound constraints: $ubc(a, b, \delta)$, where $a, b \in V, \delta \in \mathbb{N}$, imposing for the actual values $\overline{a}, \overline{b}$ of a, b that $\overline{b} \leq \overline{a} + \delta$;*
  - *lower-bound constraints: $lbc(a, b, \delta)$, where $a, b \in V, \delta \in \mathbb{N}$, imposing for the actual values $\overline{a}, \overline{b}$ of a, b that $\overline{b} \geq \overline{a} + \delta$.*

- *Ω is the process deadline, imposing that the composition ends no more than Ω time units after start.*

Basically, the activity nodes of the process model of Def. 4.3.1 are here substituted by nodes of type *call*, which represent calls to external services, whose durations range is known as we assume it is advertised by the service provider. What does the process for the Web service composition of Fig. 4.2 look like? According to Def. 4.4.1, it comprises the following elements:

- *$proc_id = example_wscomp$;*
- *$N = \{start, A, B, C, end\}$;*

- $E = \{(start, A), (A, B), (B, C), (C, end)\}$;
- $V = V^T \cup N^e$, where:
  - $V^T = \{P, Q, R\}$;
  - $N^e = \{start.s, start.e, A.s, A.e, B.s, B.e, C.s, C.e, end.s, end.e\}$;
- $C = \{d(start, 0, 0), d(A, 3, 8), d(B, 2, 6), d(C, 5, 8), d(end, 0, 0), ubc(A.s, P, 10), ubc(Q, B.e, 5), lbc(C.e, R, 6)\}$;
- $\Omega = 25$.

In Chapter 6 we address the problem of computing, at design time, adequate restrictions for the temporal parameters exchanged between Web services in a Web service composition defined after Def. 4.4.1.

4.5 Cross-Organizational Processes

A widely adopted paradigm to model and enact cross-organizational business processes [55, 73, 92, 143] is to consider them as an ensemble of local processes communicating through process views [20]. Process views abstract from the internals of the process implementation at the participating parties, and provide merely descriptions of the communication interfaces, achieving the desired low coupling between the processes, and maintain business secrecy as far as possible. A particular approach is to model cross-organizational processes by a protocol for exchanging messages governed by (Temporal) Service Level Agreements.

In such cross-organizational business processes, temporal parameters can be exchanged between the partners via messages, to communicate event timestamps without having to expose process internals in the process views. The exchange of temporal parameters enables an effective way for expressing cross-organizational temporal constraints, while preserving business secrets encoded in the local processes, and foster lean interfaces caring for loose coupling of the involved processes.

We consider here the paradigm based on send and receive communication steps, which we have addressed in [92] for the analysis and optimization of the data flow in cross-organizational processes. The process model of Def. 4.3.1 may therefore be extended with the addition of node which have types *send* and *receive* for exchang-

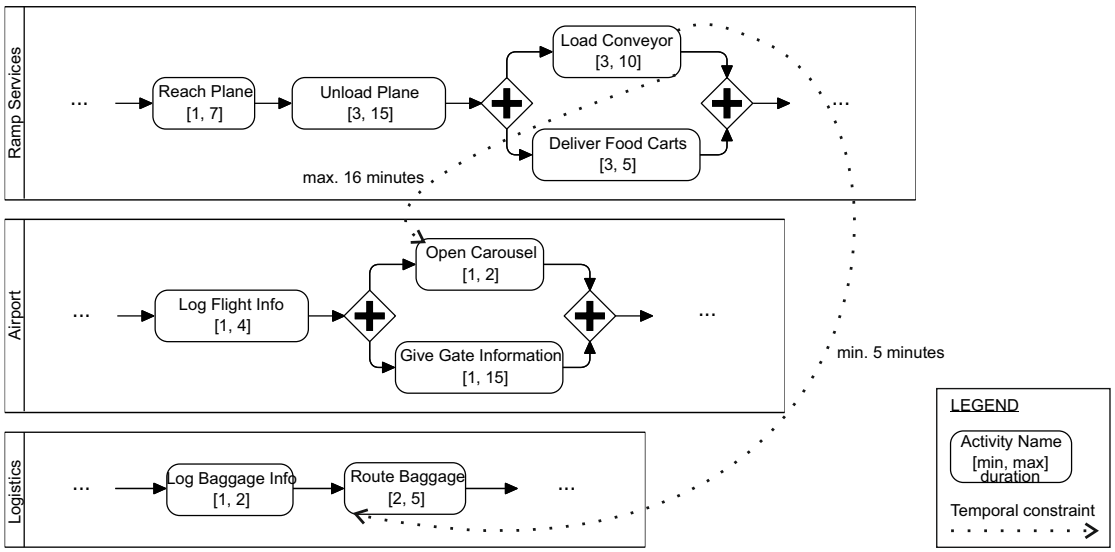

Figure 4.3: Example process in BPMN-like notation with cross-organizational constraints.

ing messages (temporal variables), and have a reference to the variables sent, resp. received, and to the receiving, resp, sending counterpart.

4.5.1 An Example for a Cross-Organizational Process with Temporal Parameters

Consider as an example the ground operations at an airport. Ground operations is an umbrella term for a large number of coordinated operations, carried out by several companies, from the time an airplane lands at an airport to the time it leaves it. Timeliness of operation in this context is crucial, since even one time failure may cause cascading ones, each with highly expensive penalty costs. For simplicity, we consider the interactions between 3 business partners (see Fig. 4.3).

After an airplane has reached the gate, the checked baggage is unloaded and brought to the terminal by a ramp services company R, then loaded into a conveyor belt system (task Lc). Each piece of baggage is routed to its destination (e.g., baggage claim area, storage area, another gate) by a logistics company L. The airport company A is responsible for the baggage claim area, where a number of carousels deliver the baggage to the passengers.

L can route the baggage not earlier than 5 minutes after *Lc*, and *A* can deliver the baggage at most 16 minutes after *Lc*. So to coordinate and execute their processes in a timely manner, *A* and *L* need to know when *Lc* ended. Additionally, to schedule and verify that their processes can be executed without time failures, they need to know in advance, the time interval in which to expect *Lc* to end. Here the end of *Lc* is an event in the local process of *R*: so how to model the cross-organizational dependencies between events of *R*, *L*, and *A*, and how to make sure they can carry on the collaboration with timeliness guarantees? Current modeling languages require to expose the internals of the collaborating processes, for expressing the time constraints required by this scenario, and for allowing a design time check for temporal correctness of the local processes and the global one. This is not desirable, because it probably exhibits business secrets and couples the process much more tightly, complicating future changes.

Similar to the above example are other application scenarios, in which there is a need for modeling (cascading) temporal requirements in cross-organizational processes. Consider, as an example, a supply chain where some raw textile materials are delivered from India to a factory in Vietnam for some first processing, and later on the processed goods need to be transferred to a second factory in Naples, Italy via cargo ship for final processing before selling. The time window for the delivery to the first factory may heavily influence the time window for fine processing and hence selling, since cargo ships may not wait in case of delays. So the various parties would require to know the possible time windows for a correct execution, while not having to disclose their processes to each other.

In the following we show how we can model the interplay of the various local temporal requirements by exchanging temporal parameters. In Chapter 7 we will argue the necessity of computing common restrictions for actual parameter values, which guarantee no time failure will occur, without the need for any of them to unnecessarily expose private information.

4.5.2 Local Process Model in a Cross-Organizational Process

We adapt the intra-organizational process model of Def. 4.3.1, to which we add two specialities to enable cross-organizational communication and enforcement of constraints: (1) *communication steps*: some of the steps send or receive messages, and

(2) *temporal parameters*: the message payload may contain input or output temporal parameters encoding temporal information.

We define the local process model for a process interacting in a cross-organizational process as follows:

Definition 4.5.1 (Local Process Model with Temporal Variables)**.** *A local process model with temporal parameters P in a cross-organizational process is a tuple $(proc_id, N, E, V, C, \Omega)$, where:*

- *$proc_id$ is a unique process id.*
- *N is the set of nodes. A node n has type $n.type \in \{start, activity, xor-split, par-split, xor-join, par-join, send, receive, end\}$. A node n is associated with two events: $n.s$ for the start, and $n.e$ for the end of the node execution.*
  - *A node n with $n.type = activity$ has a read set $n.r$ of variables read by it, and a write set $n.w$ of variables written by it.*
  - *A node ns with $n.type = send$ has an additional reference $ns.to$ to the id of the process receiving the parameter, which is in $ns.r$;*
  - *A node nr with $n.type = receive$ has an additional reference $nr.from$ to the id of the process sending the parameter, which is in $nr.w$.*
- *$E \subseteq N \times N$ is the set of edges defining precedence constraints.*
- *V is the set of temporal variables, partitioned in V^I, V^O, V^T, N^e:*
  - *V^I is the set of input parameters of P: $V^I = \bigcup_{n:n.type=receive}\{n.w\}$;*
  - *V^O is the set of output parameters of P: $V^O = \bigcup_{n:n.type=send}\{n.r\}$;*
  - *V^T is the set of local temporal variables written by some activity node: $V^T = \bigcup_{n \in N:n.type=activity}\{n.w\}$;*
  - *N^e is the set of temporal variables encoding start and end events of nodes: $N^e = \bigcup_{n \in N}\{n.s, n.e\}$.*
- *C is the set of temporal constraints consisting of*
  - *duration constraints for each $n \in N : d(n, d_{min}, d_{max})$, imposing for the actual duration $\overline{d}$ of n that $d_{min} \leq \overline{d} \leq d_{max}$;*
  - *range constraint for temporal variables $v \in V^I \cup V^O \cup V^T : r(v, v_{min}, v_{max})$, imposing for the actual value $\overline{v}$ of v that $v_{min} \leq \overline{v} \leq v_{max}$;*

- *upper-bound constraints: $ubc(a, b, \delta)$, where $a, b \in V, \delta \in \mathbb{N}$, imposing for the actual values $\overline{a}, \overline{b}$ of a, b that $\overline{b} \leq \overline{a} + \delta$;*
- *lower-bound constraints: $lbc(a, b, \delta)$, where $a, b \in V, \delta \in \mathbb{N}$, imposing for the actual values $\overline{a}, \overline{b}$ of a, b that $\overline{b} \geq \overline{a} + \delta$.*

- *Ω is the process deadline, imposing that the process ends no more than Ω time units after start.*

We highlight here the additional elements of the process model of Def. 4.5.1 to the process model of Def. 4.3.1:

1. in the set of process nodes N, there are the additional node types *send* and *receive*, and node properties $n.to$, $n.from$;
2. in the set of temporal variables V, there is the additional distinction between input and output temporal parameters, and the temporal variables which are internal to the process.

Excluding the above additions, the temporal aspect of the local process model is the same as the temporal aspect of the process model of Def. 4.3.1. It includes time-points for events, contingent durations of steps, temporal parameters, temporal constraints as relations between timepoints and/or durations, and process deadline Ω.

Temporal parameters are data elements containing a timestamp value. They can be sent to some other process via a communication channel. Input parameters are received from another process by a receiving node; output parameters are sent to another process by a sending node.

Temporal constraints restrict the points in time when events may occur. By imposing temporal constraints between parameters and events, it is possible to constrain the occurrence of local events, or events at other local processes. Constraints can only be formulated between the elements in the process model. Hence constraints between events in different processes can only be formulated, if information about the event is sent to the other process in form of a temporal parameter.

4.6 Temporal Patterns and Applications

In this section we show how typical requirements patterns can be modeled in temporally constrained processes with variables, and show how we realize the time patterns

defined in [96] by the use of temporal variables. We illustrate several of the applications of temporal variables for modeling temporally constrained processes also with some examples.

4.6.1 Dependency on External Data

1. $lbc(e, v, \delta)$: we can require that an event e occurs at least δ time units before the time specified by a temporal variable v with a lower-bound constraint.

   Example: the confirmation of attendance to an event must be sent at least two weeks before the event date specified in the invitation.

2. $lbc(v, e, \delta)$: we can require than an event e must occur at least δ time units after the time specified by a temporal variable v.

   Example: the lower-bound constraint between $last_intake_time$ and $Perform_Surgery.s$ with a bound of 24 hours in the example from Fig. 4.1.

3. $lbc(v1, v2, \delta)$: a minimum time span δ between the values assigned to 2 temporal variables can be set by imposing a lower-bound constraint between them.

   Example: a general practitioner recommends a patient to book a specialist visit at least six months after the previous visit, whose date is in the patient's record.

4. $ubc(e, v, \delta)$: we can require that an event e must occur no more than δ time units before the value specified by a temporal variable v.

   Example: a degradable product has to be shipped no more than 3 days before the date indicated in the contract by the customer for receiving it.

5. $ubc(v, e, \delta)$: we can require that an event e has to occur no later than δ time units after the timestamp specified by a temporal variable v.

   Example: a payment must be done no later than 5 days after the date specified in the invoice.

6. $ubc(v1, v2, \delta)$: we can require the maximum span between the values $v1.t$ and $v2.t$ of two temporal variables $v1$ and $v2$ to be no more than δ time units.

   Example: when advertising the important dates for a conference, the advertised date for camera ready submission must be no later than 14 days after the advertised date for acceptance notification.

It is easy to see that none of the above constraints, being based on temporal variables, can be expressed with languages which do not include temporal variables.

4.6.2 Modularized Processes, Web Service Compositions, and Cross-Organizational Processes

In the following we assume that g refers to an event in a process P, and m refers to an event in the module M included in P. We show that temporal variables can be used as input/output parameters for a module in order to express temporal constraints between events in the outer process and events in the module, and vice versa, preserving the module integrity. Figure 4.4 shows an example usage of input/output data parameters for passing temporal information between a process and a module to be used for expressing constraints.

1. $lbc(g, m, \delta)$: we can restrict the occurrence of an event m in a module to be at least δ time units after an event g in the outer process by passing the timestamp of g as a parameter (which has to be converted to the local time) to the module, and having a lower-bound constraint defined in the module.

2. $lbc(m, g, \delta)$: the occurrence of an event g in the outer process can be forced to be at least δ time units after an event m in the module with a lower-bound constraint in the outer process, by passing the value of g as an output parameter of the module, and converting it to the outer process time.

3. $ubc(g, m, \delta)$: the occurrence of an event m in the module can be restricted to be at most δ time units after the occurrence of g with an upper-bound constraint in the module referring to the (adapted to the local time) value of g passed as a module input parameter.

4. $ubc(m, g, \delta)$: we can require an event g in the global process to occur no more than δ time units after m. The value of m is an output parameter of the module.

In an analogous way, constraints between events occurring in different local processes can be converted to constraints on variables which are exchanged between these communicating local processes.

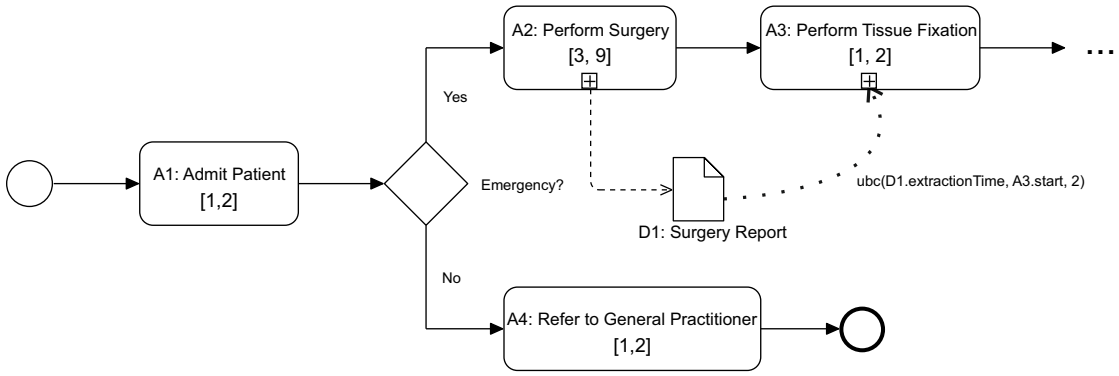

Figure 4.4: Modularized process with module time information treated as data.

4.7 Temporal Semantics

Given the formalization of the process model (and its variants), we can now define its temporal semantics. The definition applies to all process models from Def. 4.3.1, Def. 4.4.1, and Def. 4.5.1.

We define the temporal semantics of the process model by defining which executions of a process model are valid. A run, also called instance type, of a process model is a sub-graph of the process graph. Since we do not assume any knowledge about the semantics of the (control) nodes, we have no information, which of the sub-graphs are admissible in the process model. For checking the correctness of a process definition we therefore have to cope with all possible sub-graphs.

We define a scenario as a run with timestamps for the start and end events of the nodes, and for the other temporal variables:

Definition 4.7.1 (Scenario). *Let $P = (proc_id, N', E', V, C, \Omega)$ be a process, defined after Def. 4.3.1, Def. 4.4.1, or Def. 4.5.1.*

A scenario $\overline{S}(P)$ for P is a tuple (N, E, V), where $N \subseteq N', E \subseteq E', E \subseteq N \times N$. $\overline{S}(P)$ associates each $n.s, n.e \in N^e$ a timestamp $\overline{n.s}, \overline{n.e} \in \mathbb{N}$, the timepoints of the start and end events of node n in the process instance, and each $v \in V \setminus N^e$ with a value $\overline{v} \in \mathbb{N}$, the value of the temporal variable v in the scenario.

A scenario is correct, if it fulfills all temporal constraints: in such a case, we call it a valid scenario. We formalize the conditions for a scenario to be valid as follows:

$last_intake_time = 0$	$earliest_therapy_start = 39$
$start.s = 24$	$start.e = 24$
$Admit_Patient.s = 25$	$Admit_Patient.e = 26$
$Visit_Patient.s = 27$	$Visit_Patient.e = 30$
$Split.s = 31$	$Split.e = 31$
$Perform_Surgery.s = 33$	$Perform_Surgery.e = 36$
$Do_Rehabilitation.s = 40$	$Do_Rehabilitation.e = 46$
$Join.s = 46$	$Join.e = 46$
$Write_Invoice.s = 47$	$Write_Invoice.e = 48$
$end.s = 49$	$end.e = 49$

Table 4.1: Scenario for the process of Fig. 4.1.

Definition 4.7.2 (Valid Scenario). *A scenario $\bar{S}(P) = (N, E, V)$ of a process $P = (proc_id, N', E', V, C, \Omega)$ is valid, iff the following holds $\forall n, m \in N, \forall v \in V$:*

1. $0 \leq \overline{n.s}$
2. $\forall (m, n) \in E : \overline{m.e} \leq \overline{n.s}$
3. $\forall d(n, d_{min}, d_{max}) \in C : d_{min} \leq \overline{n.e} - \overline{n.s} \leq d_{max}$
4. $\forall r(v, v_{min}, v_{max}) \in C : v_{min} \leq \overline{v} \leq v_{max}$
5. $\forall ubc(a, b, \delta) \in C : \overline{b} \leq \overline{a} + \delta$
6. $\forall lbc(a, b, \delta) \in C : \overline{b} \geq \overline{a} + \delta$

With the definition of a valid scenario we formally define the semantics of temporal constraints, i.e. we define when a process instance conforms with the process definition.

We show a possible scenario for the intra-organizational process of Fig. 4.1 in Table 4.1. It is easy to verify that the scenario in the table is a valid scenario, since it meets all requirements of Def. 4.7.2.

In the following chapters we will discuss how we can verify whether a given process definition holds guarantees of yielding valid scenarios, and how to compute restrictions for temporal variables which prevent them from leading to scenarios which are not valid.

4.8 Conclusion

In this chapter we have presented a process model which features temporal variables for representing intra-organizational processes. We showed how this model can be extended to support the modeling of modularized processes, Web service compositions, and local processes which are part of a cross-organizational process. Thus, with the definitions of these process models, we have given an answer to our Research Sub-question 1.1.

We have shown examples for constraint patterns enabled by the introduction of temporal variables in the modeling capabilities of process models. We have shown that our process model and its extensions, due to the introduction of temporal variables, are able to capture a number of temporal patterns for constraints, which previous modeling languages could not model. Thus, we answered our Research Sub-question 1.2.

We defined the temporal semantics of the model through the notions of scenario and valid scenario, giving an answer to Research Sub-question 1.3.

All in all, in this chapter we have given an answer to our first Research Question. However, we did not discuss yet how to verify, whether a process definition is temporally correct, which is of interest for answering our second Research Question. In the next chapter we will show how to check the temporal correctness of process definitions defined after the proposed process models, arguing that dynamic controllability is the most adequate notion of temporal correctness of time-constrained process definitions.

Chapter 5

Checking Temporal Correctness

5.1 Introduction

In this chapter we discuss the temporal correctness of process models, as the capability of having process instances which are free from time failures, i.e. scenarios which are valid. We first motivate dynamic controllability as the most suitable notion for temporal correctness of a process model, in contrast to other existing notions. We then show how to check whether a process model is dynamically controllable, by mapping it into a temporally equivalent temporal network.

The contents of this chapter are based on the publications in [51], [62].

5.2 Motivating Dynamic Controllability

For the execution of a process the process execution system has some decisions to make. We call *process execution system* here the socio-technical system consisting of a process enactment service, a (possibly human) process manager and (possibly human) actors executing the nodes (activities and control nodes). For our purposes here we are interested in those decisions which have influence on the temporal correctness of the process execution. In our model the only decisions which can be influenced are when the execution of a node is started.

The notion of (strong) controllability of a process model [37] means that the process execution system can make these decisions *a priori* (i.e. before the process starts), in a

way that the resulting scenario is valid, regardless of external influences, i.e. the actual duration of contingent nodes, and the actual values of input temporal variables, as long as they are within their bounds. In particular, controllability demands that there is a static time plan (a fixed schedule), stating the start time of each node, such that each execution where each node starts according to the schedule is correct. Whether a process definition is controllable, depends on these static decisions made by the process execution system. However, due to the requirement of a static time plan, which precludes reacting to run time observations, controllability is frequently regarded as too strict.

A weaker condition for process temporal correctness, which has been considered in several previous works such as [147], is satisfiability. Satisfiability requires that it is possible that the execution does not raise any violation of a temporal constraint, i.e. the requirement is the existence of at least one valid scenario. However, the existence of a valid scenario does not prevent the process execution system to incur in time failures at run time. Thus, satisfiability is not a sufficient enough property for process designers, who strive for stronger guarantees.

Dynamic controllability [37] relaxes the strict notion of controllability, by allowing that the start time of a node is not fixed a priori, but may depend on the observed time-points, conditions, and durations of earlier executed nodes. A process is dynamically controllable, if the executor can apply, at run time, a dynamic execution strategy (assignment of controllable time-points), which is based on the past observations, such that the current partial scenario can be extended into a valid scenario. With dynamic controllability, the executor is given a degree of freedom to choose the assignment of time-points at run time, without the need to fix them at once before process start, as for controllability. For the increased flexibility with respect to strong controllability, and the guarantee of avoidance of time failures, we regard dynamic controllability as the most adequate notion for temporal correctness. Thus, we consider a process model to be temporally correct, if it is dynamically controllable.

5.2.1 How to Check Dynamic Controllability

Dynamic controllability of various process models has been extensively studied by several works such as [37, 82], resulting in a number of approaches to checking whether a process model is dynamically controllable.

The most widely adopted approach is based on mapping process models into equivalent temporal networks. Existing procedures for checking dynamic controllability of temporal networks, such as the procedures presented in Chapter 2, can then be applied, and the result translated to the original process model, making the dynamic controllability of a process model coincide with the dynamic controllability of its equivalent temporal network. Such an approach is preferred, in particular, due to the higher efficiency of the checking procedures when compared to, e.g., the approach based on mapping process models into Timed Gamed Automata.

Thus, here we join the position that mapping process models into equivalent temporal networks is the most suitable approach for checking dynamic controllability of process models.

To check whether a process P is dynamically controllable, it is required to know the possible durations of the process steps, the existing temporal constraints, and the possible values temporal variables may take. In particular, it is necessary to know which values the temporal variables may take, in order to check whether these values are compatible with the temporal constraints set on these temporal variables. Thus, we require that P is fully specified (see Def. 4.3.2), i.e. that ranges for all temporal variables are given. In the following of this chapter we consider only fully specified process definitions.

5.3 Mapping Process Definitions into Temporal Constraint Networks

To adequately map a process definition into an equivalent temporal network, it is necessary to establish the target temporal network, and define mapping rules which realize a correspondence of each temporal element of the process definition to an element of the temporal network. The choice on which temporal network to target depends on the acceptable trade-off between the desired expressiveness, and the complexity for checking dynamic controllability.

As discussed in Sect. 2.3.1, Simple Temporal Networks with Uncertainty (STNUs) offer efficient checking algorithms, but lack expressiveness for conditional executions. On the other hand, Conditional Simple Temporal Networks with Uncertainty (CSTNUs) cover the expression of conditionals, but at the price of higher complexity of the checking algorithms. More refined temporal constraint networks, such as the CST-

NUDs, offer even higher degrees of expressiveness, but existing approaches to check their dynamic controllability do not scale for real-life application [152]. Thus, in the following we focus on two temporal constraint networks, STNUs and CSTNUs, and show how to map process definitions into each of them for subsequent dynamic controllability check.

5.3.1 Mapping Processes into STNUs

We define here rules, which map a process model into an equivalent STNU.

Definition 5.3.1 (Mapping to STNU). *Let $P = (proc_id, N, E, V, C, \Omega)$ be a process defined as in Def. 4.3.1, Def. 4.4.1, or Def. 4.5.1. The STNU $S = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, defined as in Def. 2.3.1, equivalent to P, is obtained by applying the following rules:*

1. *Each $n \in N$ with $n.e, n.s \in N^e$ is mapped into corresponding time-points $n.e, n.s \in \mathcal{T}$;*
2. *Each $v \in V^T$ is mapped into a corresponding time-point $v \in \mathcal{T}$;*
3. *Each $e = (m, n) \in E$ is mapped into a corresponding non-contingent edge $(n.s, m.e, 0) \in \mathcal{C}$;*
4. *$(start.s, zero, 0) \in \mathcal{C}$, and $(zero, end.e, \Omega) \in \mathcal{C}$;*
5. *Each duration constraint $d(n, d_{min}, d_{max}) \in C$ is mapped into a corresponding contingent link $(n.s, d_{min}, d_{max}, n.e) \in \mathcal{L}$;*
6. *Each range constraint $r(v, v_{min}, v_{max}) \in C$ is mapped into a corresponding contingent link $(zero, v_{min}, v_{max}, v) \in \mathcal{L}$, if $v \in V^I$;*
7. *Each range constraint $r(v, v_{min}, v_{max}) \in C$ is mapped into a pair of non-contingent edges $(zero, v, v_{max}), (v, zero, -v_{min}) \in \mathcal{C}$, if $v \in V^O \cup V^T$;*
8. *Each $ubc(a, b, \delta) \in C$ is mapped into a corresponding non-contingent edge $(a, b, \delta) \in \mathcal{C}$;*
9. *Each $lbc(a, b, \delta) \in C$ is mapped into a corresponding non-contingent edge $(b, a, -\delta) \in \mathcal{C}$.*

The rules of Def. 5.3.1 take all the temporal aspects of a process model, and map them into elements of a STNU. All the time-points of the process model are mapped into time-points of the STNU with Rules 1 and 2. Rule 3 maps the (implicit) precedence constraints induced by edges of the process model into STNU non-contingent edges. Rule 4 connects the start event of the process to the *zero* time-point of the

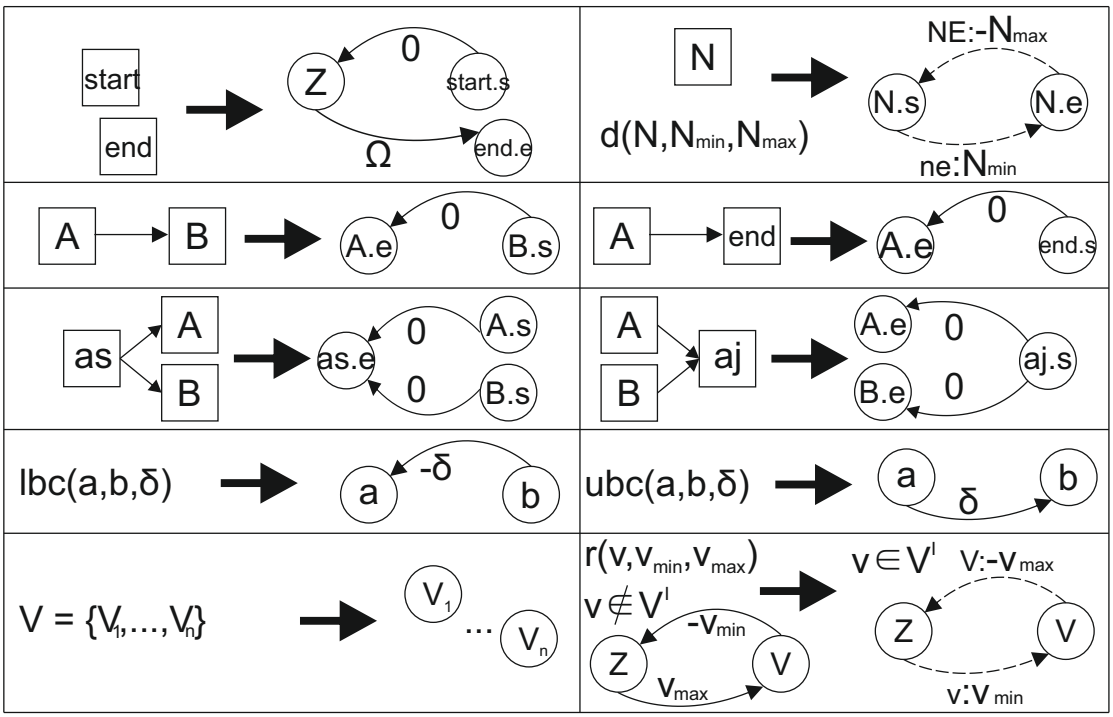

Figure 5.1: Mapping rules to map a process into a STNU.

STNU, and includes the process deadline. Rules 5 to 9 map all the explicit constraints of the process model into STNU constraints. Since we consider durations and range constraints for input parameters as contingent, durations and range constraints for input parameters are mapped into STNU contingent links (Rules 5, 6). Range constraints for other temporal variables are non-contingent, hence they are mapped into non-contingent edges (Rule 7). Upper- and lower-bound constraints are mapped into STNU non-contingent edges (Rules 8, 9).

When a process node n has a duration of 0, we simplify the resulting STNU by mapping it into a single node, making its start and end events coincide.

In Fig. 5.1 we give a graphical summary of the mapping rules pf Def. 5.3.1. In Fig. 5.2 we show the STNU obtained by applying the mapping rules of Def. 5.3.1 to the process depicted in Fig. 4.1. In the STNU, 0-durations are collapsed into single nodes.

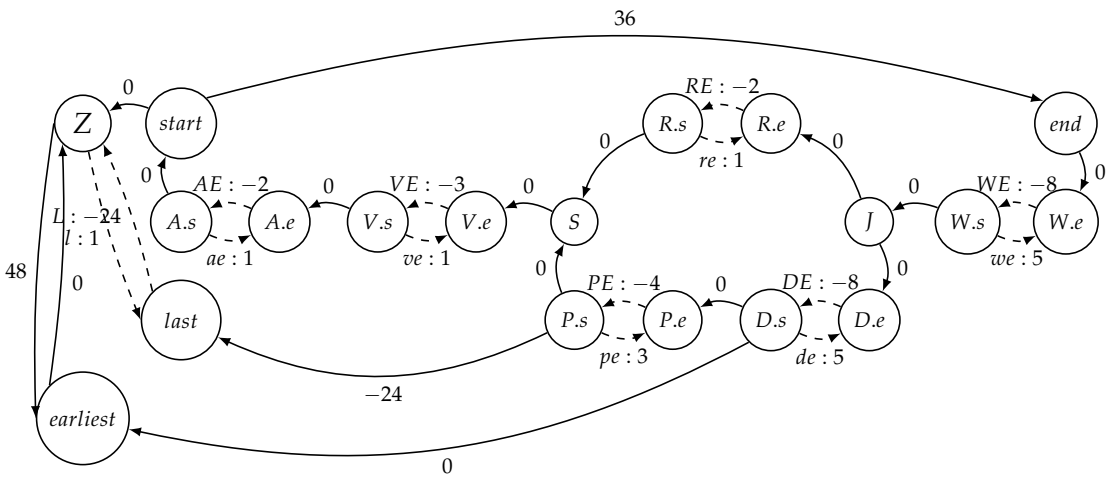

Figure 5.2: STNU equivalent to the process of Figure 4.1 obtained with the application of the mapping rules of Def. 5.3.1. The graphical notation follows the notation of [116]. For a compact presentation, the initials of the process nodes are used for labeling the corresponding STNU nodes.

5.3.2 Mapping Processes into CSTNUs

We now define rules, which map a process model into an equivalent Conditional Simple Temporal Network with Uncertainty (CSTNU).

Definition 5.3.2 (Mapping to CSTNU)**.** *Let $P = (proc_id, N, E, V, C, \Omega)$ be a process defined as in Def. 4.3.1, Def. 4.4.1, or Def. 4.5.1. The CSTNU $S = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, defined as in Def. 2.3.3, equivalent to P, is obtained by applying the following rules:*

1. *Each $n \in N$ with $n.type \neq xor - split$ and $n.e, n.s \in N^e$ is mapped into corresponding time-points $n.e, n.s \in \mathcal{T}$;*

2. *Each $n \in N$ with $n.type = xor - split$ and $n.cond = p$ and $n.e, n.s \in N^e$ is mapped into corresponding time-points $n.s, n.e \in \mathcal{T}$, with $n.e$ observation node with observed condition p;*

3. *Each $v \in V^T$ is mapped into a corresponding time-point $v \in \mathcal{T}$;*

4. *Each $e = (m, n) \in E$ is mapped into a corresponding non-contingent edge $(n.s, m.e, 0, e.cond) \in \mathcal{C}$;*

5. *$(start.s, zero, 0, True) \in \mathcal{C}$, and $(zero, end.e, \Omega, True) \in \mathcal{C}$;*

6. *Each duration constraint* $d(n, d_{min}, d_{max}) \in C$ *is mapped into a corresponding contingent link* $(n.s, d_{min}, d_{max}, n.e, True) \in \mathcal{L}$;

7. *Each range constraint* $r(v, t_{min}, t_{max}) \in C$ *is mapped into a corresponding contingent link* $(zero, t_{min}, t_{max}, v, True) \in \mathcal{L}$, *if* $v \in V^I$;

8. *Each range constraint* $r(v, v_{min}, v_{max}) \in C$ *is mapped into a pair of non-contingent edges* $(zero, v, v_{max}, True), (v, zero, -v_{min}, True) \in \mathcal{C}$, *if* $v \in V^O \cup V^T$;

9. *Each* $ubc(a, b, \delta) \in C$ *is mapped into a corresponding non-contingent edge* $(a, b, \delta, True) \in \mathcal{C}$;

10. *Each* $lbc(a, b, \delta) \in C$ *is mapped into a corresponding non-contingent edge* $(b, a, -\delta, True) \in \mathcal{C}$.

The rules of Def. 5.3.2 produce the same effect as the rules of Def. 5.3.1, with the addition of generating CSTNU observation nodes, in which xor-conditions are observed, and edges which are labeled with the observed conditions.

In Fig. 5.3 we give a graphical summary of the mapping rules of Def. 5.3.2. For a compact representation, we left implicit the depiction of the empty labels.

The correctness of the mapping is shown in the following Lemma:

Lemma 5.3.1. *The mapping of a process P into the STNU, resp. CSTNU S, resulting from the application of the mapping rules of Def. 5.3.1, resp. Def. 5.3.2 is correct.*

Proof. The correctness of the mapping can easily be shown, as there is a bijective mapping of the time points in the scenarios of P to the time points of S. Additionally, S encodes exactly the set of inequalities between these time points as defined in Def. 4.7.2 of valid scenario (inequalities for precedence constraints, durations constraints, range constraints, and upper- and lower-bound constraints). □

5.4 Checking Dynamic Controllability

Once a fully specified process model P is mapped into a temporal network S, existing procedures developed for checking the dynamic controllability of such a temporal networks, as described in Sect. 2.3.1, can be applied to S. Due to the correctness of the mapping, the result of the dynamic controllability check on S also applies to P:

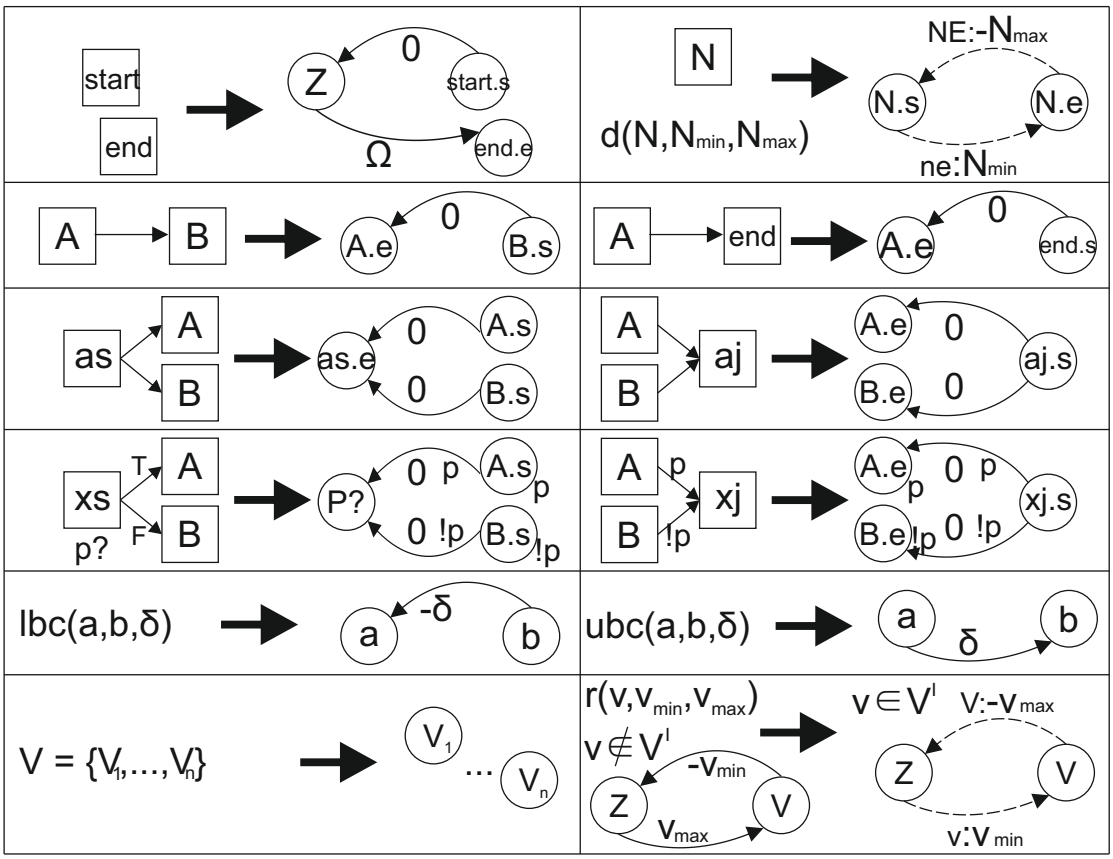

Figure 5.3: Mapping rules to map a process into a CSTNU.

Theorem 5.4.1 (Dynamic Controllability Equivalence)**.** *Let $P = (proc_id, N, E, V, C, \Omega)$ be a fully specified process definition. Let S be the temporal network equivalent to P, obtained from the application of the mapping rules of Def. 5.3.1 or Def. 5.3.2. Then P is dynamically controllable if and only if S is dynamically controllable.*

Proof. The proof directly follows from the correctness of the mappings of Lemma 5.3.1. □

Thus, it is possible to design a procedure which takes as input a process definition P and a target temporal network (either STNU or CSTNU). The corresponding mapping rules from Def. 5.3.1, resp. Def. 5.3.2 are then applied to P, deriving an equivalent STNU, resp. CSTNU S. Dynamic controllability checking procedures are

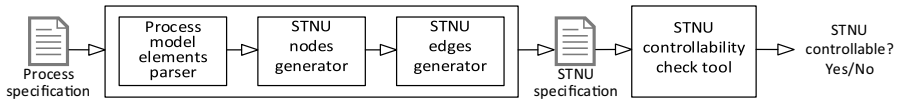

Figure 5.4: Mapping application architecture pipelined with STNU controllability check tool.

applied to S, and the result is transferred to P. In the following section we describe an implementation for such an approach.

As shown in Chapter 2, formal proofs of complexity indicate that the existing algorithms for checking dynamic controllability of STNUs scale better than algorithms for CSTNUs. Such a difference follows from the fact that CSTNUs offer higher expressiveness, in allowing the distinction between different execution paths which depend on observed conditions. Consequently, checking whether a process is dynamically controllable by mapping it into a CSTNU requires exploring all possible execution paths, which are exponential in the number of xor-splits. On the other hand, checking the dynamic controllability of a process by mapping it into a STNU considers all constraints as belonging to a single execution path. Thus, an approach based on mapping the process into a STNU results in a more efficient checking procedure, but at the same time the dynamic controllability result may be stricter than the result which a CSTNU-based approach would provide.

For the process of Fig. 4.1, the application of constraint propagation procedures to its equivalent STNU shown in Fig. 5.2, returns `True`, meaning that the process is dynamically controllable.

5.5 Implementation

As a proof of concept, we have implemented a prototype application which, given a process definition compliant with Def. 4.3.1 and a target temporal constraints network (STNU or CSTNU), applies an implementation of the rules of Def.5.3.1, resp. Def.5.3.2, producing a temporal constraint network S. Existing algorithms for dynamic controllability check can be applied to S to check the dynamic controllability of the process definition.

Fig. 5.4 shows the architecture of the prototype. It consists of three modules: one for the parsing of process model elements, one for generating STNU/CSTNU nodes,

# Nodes	# Constraints	# Variables	Min. Time	Max. Time	Average Time
15	7	3	9ms	10ms	11ms
30	15	7	21ms	30ms	25ms
60	30	15	34ms	45ms	37ms
120	60	30	55ms	70ms	60ms

Table 5.1: Measured times for mapping process definitions into temporally equivalent Temporal Constraint Networks.

and one for generating STNU/CSTNU edges. First the input file, containing the process definition, is parsed. The read information is used to generate STNU/CSTNU nodes out of process nodes and variables. Finally, the generated nodes are connected with edges corresponding to the constraints identified in the parsing phase. The resulting temporal network is stored in a file format compliant with the format which existing frameworks such as CSTNUEditor[1] can read as input for visualization and dynamic controllability check.

Experimental results, based on the execution of the mapping application on process models of various sizes on a Windows 10 physical machine with 16GB RAM and i7 CPU, showed the feasibility of the mapping strategy. We report the results in Table 5.1. The implementation scales well, even in the case of randomly generated process definitions with 120 nodes, 30 temporal variables, and 60 temporal constraints of various types, producing equivalent STNUs or CSTNUs in 60ms on average, indicating that the process model size, and the target temporal network, do not impact the applicability of the mapping rules.

Additionally, we have used CSTNUEditor to perform dynamic controllability checks on the temporal networks generated with the mapping application. The checking tool takes a temporal network as input, and determines whether it is dynamically controllable or not, by applying constraint propagation methods [82]. The measured performance in our experiments is in line with the results found in [16], where empirical proof of feasibility for checking the dynamic controllability of networks of various sizes using the checking tool has been shown.

Pipelining our implementation with existing tools for dynamic controllability check enables the assessment of temporal qualities of process definitions defined after Def. 4.3.1. In particular, we show that process designers have now a complete framework that enables: 1) the definition of process models with temporal variables, and 2)

[1] `http://profs.scienze.univr.it/~posenato/software/cstnu/`

the verification of the dynamic controllability of such process models at design time when they are fully specified. Thus, the framework allows us to pro-actively help designers and process managers in refining or correcting design choices, in particular for what concerns the temporal aspects of a process, without the need to gather process execution information.

What is the computational complexity for determining whether a process model is dynamically controllable? The mapping procedure is linear in the number of process nodes, variables, edges, and constraints. Thus, the equivalent STNU/CSTNU can be generated with a single parse of the process specification. The dominating cost is given by the dynamic controllability checking algorithm applied, which we discussed in Sect. 2.3.1.

The experimental results support in particular the application of constraint propagation algorithms to the STNUs obtained from the mapping, due to the lower complexity with respect to CSTNUs. Further, the results encourage interpreting additional information derived from constraint propagation from the point of view of process definitions, as will be shown in the next chapter.

5.6 Conclusion

In this chapter we have addressed the problem of verifying whether a process definition allows for process instances free from time failures, and defined such a capability as the temporal correctness of a process.

In particular, we have motivated dynamic controllability as the most suitable notion for temporal correctness, thus answering our Research Sub-question 2.1.

We have shown how that mapping process models into temporally equivalent Temporal Constraint Networks is a viable approach for checking the dynamic controllability of these process models, thus giving an answer to Research Sub-question 2.2.

We have presented two approaches to map a process definition into equivalent temporal constraint networks, to enable checking dynamic controllability. The check is based on existing checking algorithms, designed for Temporal Constraint Networks. We have in particular discussed STNUs and CSTNUs as the two most adequate temporal networks target of the mapping, providing the definition of mapping rules, thus answering Research Sub-questions 2.3 and 2.4.

We have additionally presented an implementation of an application, which takes a process definition as input, and transforms it into an equivalent temporal constraint networks of user's choice. The generated temporal constraint network can be verified for dynamic controllability using existing tools.

In this chapter we have therefore provided a comprehensive answer to our Research Question 2, on how to verify the temporal correctness of a process model featuring temporal variables.

So far we have considered process definitions which are fully specified, i.e. in which all temporal variables have specified ranges for their allowed values. In the next chapter we address the problem of computing missing range constraints for temporal variables, which yield guarantees of dynamic controllability for the process in which they are used.

Chapter 6

Computing Restrictions for Temporal Parameters

6.1 Introduction

In Chapter 4 we have argued that the process model of Def. 4.3.1 may be used to represent also modularized processes. Similarly, the model of Def. 4.4.1 can capture Web service compositions, and the model of Def. 4.5.1 local processes participating into a cross-organizational process. In these process models temporal parameters are used to establish temporal constraints to realize Temporal SLAs.

For advertising such Temporal SLAs it is crucial to publish which values for parameters can be accepted, e.g., which deadlines or milestones can be guaranteed, which requires that the service provider knows, which ranges of values are admissible for the temporal parameters. These ranges are the ranges in which temporal parameters are allowed to take values, without risking a violation of Temporal SLAs under any foreseeable imponderabilities.

Which are the imponderabilities that may lead to temporal violations? The major source of such uncertainty is the duration of activities and service calls, when their duration is limited by a minimum and maximum duration, but the actual value can only be observed rather than controlled. We have shown in the previous chapters that such durations are contingent durations.

Now, on the one hand, a process model has a degree of uncertainty, due to the contingent durations. On the other hand, a process designer needs to make sure that

all temporal requirements are met, despite such uncertainty. For processes which can be instantiated with a number of temporal parameters, there is additional uncertainty in the actual values for these parameters. Nevertheless, for such processes, an execution which meets all temporal constraints must be ensured, despite contingencies and actual values for parameters passed at instantiation.

In Chapter 5 we have addressed the problem of checking whether a fully specified process definition is dynamically controllable, i.e. when all temporal variables have a specified range. However, it is not always the case, that such a range is known.

In this chapter we address the question: how to restrict the admissible values for input temporal parameters, such that temporal correctness can be guaranteed under any circumstance? These restrictions form a set of necessary and sufficient conditions, which would allow a service provider or process designer to include these conditions when advertising the Web service composition, resp. the process, and its acceptable T-SLAs. In the following we define a sound and complete procedure to derive the most relaxed set of conditions on input temporal parameters to guarantee an execution of a Web service composition, resp. a process, without time failures.

The contents of this chapter are based on the publications in [63], [64], [66].

6.2 Computing Input Temporal Parameter Restrictions

To motivate the need for computing parameter restrictions with an example, consider the simple Web service composition shown in Fig. 4.2. One can verify (see Table 6.1) that invoking this composition with a parameter instantiation $\langle P, Q, R \rangle = \langle 13, 9, 28 \rangle$ may result in a violation of the temporal constraints, due to the uncontrollable nature of the durations of A, B, and C. On the other hand, a parameter instantiation $\langle P, Q, R \rangle = \langle 10, 9, 28 \rangle$ would guarantee that, no matter how long the service calls turn out to take, it is always possible to schedule them such that all temporal constraints are met. How can we ensure that only parameter instantiations allowing the fulfillment of all temporal constraints are possible?

We consider actual values of input parameters which are set at instantiation time. These values must be compatible with the existing constraints and the contingent durations of service calls observed at run time. This is necessary to avoid any possible time failures (i.e. missing any temporal constraint) at run time.

Event	Timestamp	Constraint	Constraint met?
start	0	none	
A.start	3	$P \leq A.start + 10$	YES
A.end	11	none	
B.start	11	none	
B.end	17	$B.end \leq Q + 5$	NO
C.start	17	none	
C.end	25	$R \geq C.end + 6$	NO
end	25	$end \leq start + 25$	YES

Table 6.1: Example execution times for the composition in Fig. 4.2, with $P = 13$, $Q = 9$, $R = 28$.

However, it is not possible to predict: (1) the actual values of parameters, as they are given at instantiation time, and (2) the actual durations of service calls, as they are observed at run time. We nevertheless show in the following that it is possible, at design time, to compute adequate restrictions on the parameter values, which give dynamic controllability guarantees.

What do we mean by adequate restrictions? If the parameter ranges are too small, possible applications and clients are lost; if they are too broad, however, they could cause penalty costs for violating temporal constraints. So by adequate parameter restriction we mean the broadest parameter restriction, i.e. constraint, which does not violate dynamic controllability:

Definition 6.2.1 (Broadest Parameter Restriction)**.** *Let S be a dynamically controllable STNU. Let P, Q be nodes in S. We call a restriction $c = (P \leq Q + \delta)$ in S the broadest, if $\nexists$ restriction $c' = (P \leq Q + \delta')$ with $\delta < \delta'$ in S, such that $c \implies c'$.*

Further, we call broadest range restriction for a parameter P the pair of broadest restrictions on P which limit its minimum and maximum values. We propose a technique which computes the broadest ranges for temporal parameters which guarantee that all temporal constraints are fulfilled, independently of uncertainties in the execution of the process.

We consider the STNU equivalent of a process model with input parameters, obtained by applying the mapping rules of Def. 5.3.1, which is verified to be dynamically controllable. We apply to it STNU constraint propagation rules, which make redundant all constraints on parameters, which depend on contingent durations. In particular, with these constraint propagation rules we derive non-contingent constraints, which relate parameters only to the STNU *zero*, and to other parameters when they

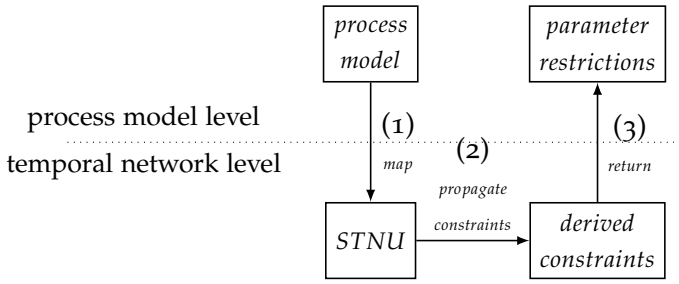

Figure 6.1: Overview of the approach for computing parameter restrictions.

depend on each other. These derived constraints represent the parameter restrictions (minimum and maximum allowed values, inter-dependencies) which need to be fulfilled for the STNU to be dynamically controllable.

These constraints are translated back to the level of the process model as the restrictions on the input parameters, which need to be fulfilled at instantiation time. These restrictions are such that they permit temporal parameters to take only values, at process instantiation, which allow for dynamic controllability, independent from any subsequently observed contingent duration. Thus, being computed at design time, they may be more strict than if they were computed at run time, after observing actual contingent durations. We sketch the approach in Fig. 6.1.

6.3 Constraint Propagation Rules

We propose a set of new STNU constraint propagation rules, which aim at finding the broadest restrictions for parameters, which hold dynamic controllability guarantees in a dynamically controllable STNU S. The constraint propagation rules we propose are an adaptation of the 3 rules of the RUL system [17].

In the following we assume that the STNU to which the rules are applied is already verified to be dynamically controllable.

P-Relax

The $P - Relax$ rule derives a non-contingent constraint from two existing non-contingent constraints (see Fig. 6.2). It is defined exactly as *Relax* rule of the RUL system [17].

Definition 6.3.1 (P-Relax). *Let v be the weight of a non-contingent edge from time point P to time point Q in a STNU S; let w be the weight of a non-contingent edge from time point Q to time point R. Then a new non-contingent edge from P to R can be derived, with weight $v + w$.*

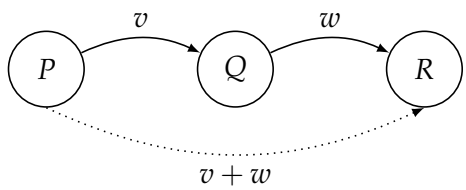

Figure 6.2: P-Relax rule. The derived constraint is dotted.

Example Suppose that P, Q, and R in Fig. 6.2 represent events *Hold exam*, *Communicate exam results*, and *Confirm exam grades*, respectively. If the results must be communicated at most 21 days after the exam took place ($v = 21$), and the grades must be confirmed no more than 14 days after they have been communicated to the students ($w = 14$), it follows that confirming the grades has to be done no later than $21 + 14 = 35$ days after the exam date.

Lemma 6.3.1. P-Relax *derives the broadest non-contingent constraint between time point P and time point R.*

Proof. Since *P-Relax* is defined as *Relax* in the RUL system, for the proof we refer to [17]. □

P-Lower

The $P - Lower$ rule derives a non-contingent constraint from an existing contingent link and a non-contingent constraint coming from the contingent time point of the link (see Fig. 6.3).

Definition 6.3.2 (P-Lower). *Let (A^C, l, u, C) be a contingent link between A^C and C in a STNU S; let w be the weight of a non-contingent edge from C to a time point P. Then a new non-contingent edge from A^C to P can be derived, with weight $w + l$.*

Example Suppose that A^C and C in Fig. 6.3 are the call and return events of a Web service *Gather hotel offers*, which takes 1 to 3 minutes. Parameter P represents the time until which a 10% discount on the hotel price is offered: no more than 60 minutes after

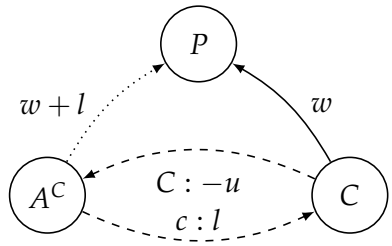

Figure 6.3: P-Lower rule. The derived constraint is dotted.

the service call returns ($w = 60$). Then it follows that discount must be valid for no more than 61 minutes after the service is called.

Lemma 6.3.2. P-Lower *derives the broadest non-contingent constraint* $c = (P \leq A^C + w + l)$ *between time point* A^C *and time point* P *in a dynamically controllable STNU* S*, which makes the constraint* $(P \leq C + w)$ *between* C *and* P *redundant.*

Proof. For the redundancy, it is easy to see that the introduction of c in S requires to satisfy $P \leq A^C + w + l$. $\forall d : l \leq d \leq u, A^C + d \leq C$. So $P \leq A^C + w + l \implies P \leq C + w$.

For the broadness: suppose that the derived constraint between A^C and P is any less strict, e.g., $P \leq A^C + w + l + \epsilon$ (where for our process models we can consider $\epsilon = 1$, since time is measured with integer values). Then in the scenario in which $P = A^C + w + l + \epsilon$ and the contingent link takes its minimum duration ($C = A^C + l$), the requirement that $P \leq C + w$ from the original constraint is violated by (at least) ϵ time units. Therefore $P \leq A^C + w + l$ is the most lenient constraint which can be set between A^C and P without contradicting the constraint between C and P. □

P-Upper

The $P - Upper$ rule derives a non-contingent constraint from an existing contingent link and a non-contingent constraint from a time point P to the contingent time point of the link (see Fig. 6.4).

Definition 6.3.3 (P-Upper)**.** *Let* (A^C, l, u, C) *be a contingent link between* A^C *and* C *in a STNU* S*; let* w *be the weight of a non-contingent edge from a time point* P *to time point* C*. Then a new non-contingent edge from* P *to* A^C *can be derived, with weight* $w - u$*.*

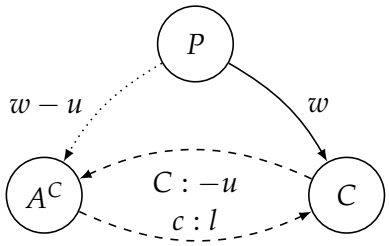

Figure 6.4: P-Upper rule. The derived constraint is dotted.

Example Suppose that A^C and C in Fig. 6.4 are call and return events of a Web service *Process payment*, which takes 1 to 5 minutes. Parameter P represents the last time of validity of the price to be paid: the payment must be completed at most 15 minutes after it ($w = 15$). Then it follows that the payment service must be called no later than $15 - 5 = 10$ minutes after the time specified in P.

Lemma 6.3.3. P-Upper *derives the broadest non-contingent constraint* $c = (A^C \leq P + w - u)$ *between time point* P *and time point* A^C *in a dynamically controllable STNU* S*, which makes the constraint* $(C \leq P + w)$ *between* P *and* C *redundant.*

Proof. For the redundancy, it is easy to see that the introduction of c in S requires to satisfy $A^C \leq P + w - u$, i.e. $A^C + u \leq P + w$. $\forall d : l \leq d \leq u, A^C + d \leq C$. So $A^C \leq P + w - u \implies C \leq P + w$.

For the broadness: suppose that the derived constraint between P and A^C is any less strict, e.g., $A^C \leq P + w - u + \epsilon$. Then the scenario in which $P = A^C - w + u - \epsilon$ and the contingent link takes its maximum duration ($C = A^C + u$), the original constraint $C \leq P + w$ is violated by (at least) ϵ time units. Therefore $A^C \leq P + w - u$ is the most lenient constraint which can be set between P and A^C without contradicting the constraint between P and C. □

With the constraint propagation rules defined, we can now design an algorithm, which iteratively applies these rules to derive restrictions for the parameters of a Web service composition. In particular, we are interested in deriving constraints from the *zero* time point to each parameter P_i, and vice versa, which state its maximum and minimum allowed values; and between pairs of parameters, if they exist.

6.4 An Algorithm to Derive Parameter Restrictions

The algorithm is defined in Alg. 6.1. It takes a process P as input, and returns a set of restrictions for the process input parameters.

The first step (line 2) is mapping the process definition into an equivalent STNU S, according to the mapping rules shown in Chapter 5.

Then, the STNU is checked for dynamic controllability using any existing checking procedure (line 3). If S is not dynamically controllable, the algorithm terminates with an error (*not_dc*, line 4). In such a case, the process definition needs to be redesigned in order to make it dynamically controllable.

If S is dynamically controllable, then there exists at least one assignment of values for its input parameters which complies with the dynamic controllability of the process. So a complete propagation of the constraints is performed, using the rules defined above (lines 6-11). When any rule R is applicable, then R is applied to S. The STNU is then checked again for dynamic controllability, since the new edge may violate such a property (line 8). Such a violation is possible, when a parameter value cannot be fixed at instantiation time, but has to be compatible with some run time observation. The propagation stops when network quiescence is reached, i.e. no rule can be applied anymore.

Finally, a STNU S' composed of the set N of time points for parameters and time point *zero* (function *selectParameters* in line 13), and of all the constraints C between them (function *selectParameterConstraints* in line 14), is returned (line 15).

6.4.1 Interpretation of the Results

How do we interpret these derived constraints? An edge from node *zero* to the node for a parameter P with weight p_{max} indicates the maximum value p_{max} which P is allowed to take; an edge from P to *zero* with weight $-p_{min}$ indicates the minimum value p_{min} which P is allowed to take. The absence of an edge from *zero* to P or of an edge from P to *zero* indicates that there is no maximum, resp. minimum bound for the value of P. An edge between any two parameter nodes P_i and P_j indicates a restriction relating their values which needs to be fulfilled in addition, such as a maximum distance between the value of P_i and the value of P_j.

The result of the application of Alg. 6.1 to the STNU from the running example, showing the derived restrictions for the parameters, is depicted in Fig. 6.5. The computed requirements are that P has to be at most 13, Q at least 9, and R at least 28 after start; additionally, P is at most 1 after Q, and at least 18 before R. We can compare the parameter instantiations shown in the example of Fig. 4.2 against these derived restrictions. It is easy to see, that the valid instantiation $\langle P, Q, R\rangle = \langle 10, 9, 28\rangle$ is compatible with the derived restrictions. On the other hand, $\langle P, Q, R\rangle = \langle 13, 9, 28\rangle$ does not meet all requirements, and it is shown to lead to a constraint violation at run time (see Table 6.1).

Any instantiation of the process, in which the actual parameters fulfil all the restrictions, is guaranteed to be carried out without any time failure.

6.4.2 Soundness and Completeness of the Algorithm

We can now show that the algorithm realizes a sound and complete procedure for computing the broadest parameter restrictions which hold dynamic controllability guarantees.

Theorem 6.4.1. *Alg. 6.1 terminates, and it is a sound and complete procedure to derive the broadest restrictions for the temporal parameters of a process, which guarantee dynamic controllability of the process.*

Algorithm 6.1 Compute parameter restrictions

```
1:  Input: P
2:  S := mapToSTNU(P)
3:  if (¬isDynamicallyControllable(S)) then
4:     return not_dc
5:  else
6:     while ((R := isRuleApplicable(S)) ≠ null) do
7:        S := applyRule(R, S)
8:        if (¬isDynamicallyControllable(S)) then
9:           return not_dc
10:       end if
11:    end while
12: end if
13: N := selectParameters(S, P) ∪ {Z}
14: C := selectParameterConstraints(S, N)
15: return S′(N, C)
```

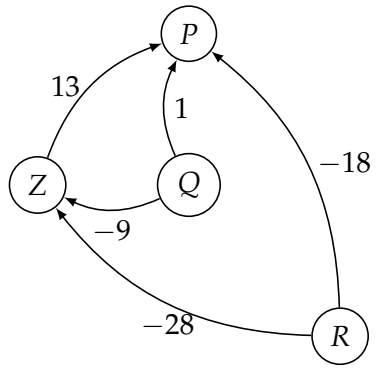

Figure 6.5: Restrictions for the parameters of the Web service composition of Fig. 4.2 as STNU constraints, obtained from the application of Alg. 6.1.

Proof. Termination follows from the facts that the values for the timestamps are bounded (overall deadline, *zero*), their domain is the set of natural numbers, and that the rules derive tighter constraints, thus reducing the set of possible solutions. Thus, the procedure reaches a fix-point (solution or contradiction through a negative loop) after a finite number of rule applications.

To show that the procedure is sound, it is sufficient to show that each of the rules produces an edge representing a constraint, which is at least as strict as the constraint of the edge deriving from the corresponding rule of the RUL system. The RUL system can be employed to check the dynamic controllability of a STNU, by propagating constraints based on the 3 rules.

The procedure is complete, in the sense that it computes the broadest restrictions for the parameter values. This follows from the fact that with the application of the rules of Def. 6.3.1, 6.3.2, 6.3.3, the broadest constraints are derived, as shown in the proofs of Lemmas 6.3.1, 6.3.2, 6.3.3, and no applicable rule is made inapplicable by any other rule application. □

Given the theorem of soundness and completeness of the proposed algorithm, we can evaluate the feasibility and scalability of the approach by implementing Alg. 6.1 and executing it on a number of cases.

6.5 Implementation

As a proof of concept for the practical applicability of the algorithm, we have implemented a Java application which realizes the pseudo-code shown in Alg. 6.1, and measured its execution times on a number of processes representing Web service compositions.

The application takes as input a timed graph with parameters, and produces an equivalent STNU by applying a mapping procedure based on the rules shown in Chapter 5. The resulting STNU is checked for dynamic controllability using an implementation of the constraint propagation based on the RUL system [17]. If the checking procedure returns `True` (the STNU is dynamically controllable), the loop of constraint propagation using the proposed rules is applied. If the loop terminates successfully, the application returns a STNU composed only of *zero*, the parameters, and their respective constraints, from which corresponding restrictions are returned as inequalities.

For the experimental setup, we run our experiments on a Windows 10 physical machine, with i7 CPU and 16GB of memory. We considered dynamically controllable processes of varying size, having a number of activity nodes ranging from 5 to 30, and number of parameters ranging from 1 to 12. For each process size and number of parameters, we randomly generated 25 instances, and verified that they were all dynamically controllable, so that the algorithm would not immediately terminate without applying any rule.

In Table 6.2 we report the minimum, maximum, and average times measured for executing the algorithm implementation. We observed a nonlinear growth of the measured average execution times, which is due to checking for dynamic controllability after each application of a constraint propagation rule. However, for large process sizes, with 30 activity nodes and 12 temporal parameters, the average time needed for computing parameter restrictions is less than 23 minutes, which we deem acceptable, considering that it is a design time task and has to be performed only once. With the support of our experimental results, we regard the proposed algorithm as practically applicable for real life situations.

# Activity Nodes	# Parameters	Min. Time	Max. Time	Average Time
5	1	0.00s	0.09s	0.01s
5	3	0.02s	0.55s	0.22s
10	3	0.02s	5.08s	2.25s
10	6	1.00s	20.21s	5.35s
20	6	0.84s	317.77s	113.63s
20	9	11.34s	423.60s	149.91s
30	9	273.22s	3190.14s	1207.00s
30	12	266.09s	2873.81s	1353.79s

Table 6.2: Results of the experimental evaluation for Alg. 6.1.

6.6 An Alternative Approach to Compute Missing Ranges

In this section we present an alternative approach for deriving ranges for input temporal parameters, that are necessary and sufficient restrictions on these parameters for the dynamic controllability of a process. To compute such ranges, we make use of a technique for deriving constraints in a CSTNU, based on piecewise-linear functions (PLFs) [81]. We use in particular an operation this new technique offers.

Given a CSTNU, which is dynamically controllable, and 2 time points X and Y, the technique allows us to derive the strongest non-contingent constraint $X - Y \leq \delta$ which can be added to the network without losing the dynamic controllability property.

This means for any $\delta' < \delta$ the network will not be dynamically controllable. It also means that we can add a non-contingent constraint $X - Y \leq \delta''$, if $\delta < \delta''$.

In the following we abstract this operation in the following way and package it into the operation *scon* (strongest *con*straint):

Definition 6.6.1 (scon)**.** *Let S be a CSTNU which is dc; let X and Y be timepoints in S. The function $scon(S, X, Y)$ yields the smallest δ, such that the non-contingent constraint $X - Y \leq \delta$ is such that $S \cup \{(Y, X, \delta)\}$ is dc.*

The technique introduced in [81] replaces the usual numerical weights of CSTNU edges with PLF functions of a variable δ, and applies extended constraint propagation rules to determine *scon* by solving δ.

6.6.1 A Necessary and Sufficient Range for an Input Parameter

We exploit the operation *scon* for finding strongest constraints of [81] to compute the range for an input parameter in a process definition, which can be exposed as a range constraint. The procedure works as follows.

Let S be the CSTNU for the process P with parameter X. We compute $u = scon(S, X, Z)$. This means that u is the smallest value δ can assume such that including $X - Z \leq \delta$ maintains the dynamic controllability, i.e. S is not dynamically controllable, if $X < u$.

Then we compute $l = -scon(S, Z, X)$. This means that $-l$ is the smallest value for δ such that including $Z - X \leq \delta$ maintains S dynamically controllable. This means that S is not dynamically controllable, if $X > l$.

Now we must distinguish 3 cases:

1. $l < u$: In this case the network tells us that S is dynamically controllable, and for each possible run there exist an instantiation x for X with $l \leq x \leq u$. However, as it is not possible to shrink the interval any further, the assignment of a value for X is dynamic, and depends on observations but cannot be done at process start. This means the process is not dynamically controllable for any actual input parameter.

2. $l = u$: In this case S is *dc* for the value $X = u = l$ and we can add two non-contingent links (Z, X, u) and $(X, Z, -u)$ to S, which is equivalent to adding a contingent link (Z, u, l, X) to S, which would correspond to a narrow range constraint of $[u, u]$ for X in the process definition. However, $[u, u]$ is not a proper range for an input parameter, since it would force the parameter to take only a value u.

3. $u < l$: In this case we can add a contingent link (Z, u, l, X) to S, which corresponds to a range constraint of $[u, l]$ for X in the process definition.

Theorem 6.6.1. *Let X be a parameter in a process P, S be the CSTNU for P, with S dynamically controllable. Let $l = -scon(S, Z, X)$ and $u = scon(S, X, Z)$. Then:*
(1) $[u, l]$ is a necessary and sufficient range condition for X such that P is dynamically controllable.
(2) $S \cup \{(Z, u, l, X)\}$ is dynamically controllable.

Proof. Necessary condition: from the definition of strongest constraint [81] we conclude that $u = scon(S, X, Z)$ is the smallest value δ can assume such that including $X - Z \leq \delta$ maintains the dynamic controllability of P. This means that S is not dynamically controllable, if $X < u$. On the other side, if $l = -scon(S, Z, X)$, then $-l$ is the smallest value for δ such that including $Z - X \leq \delta$ maintains S dynamically controllable. Hence S is not dynamically controllable, if $X > l$. Combined, we conclude that $X \in [u, l]$ is a necessary condition for S, and hence P, being dynamically controllable.

Sufficient condition: Let $x \in [u, l]$. From $u = scon(S, X, Z)$ follows that adding $X - Z \leq x$ maintains S dynamically controllable; from $l = -scon(S, Z, X)$ follows that including $Z - X \leq x$ maintains S dynamically controllable. Combined, we derive that S remains dynamically controllable, if we require $X = x$. Since this holds for all $x \in [u, l]$, the range constraint $[u, l]$ is a sufficient condition that S, and hence P, is dynamically controllable.

Contingent link: Following the argumentation above, for each $x \in [u, l]$ we can add non-contingent links (Z, X, x) and $(X, Z, -x)$ and maintain the dynamic controllability, by fixing the parameter X to the value x. Since we can do this for all $x \in [u, l]$, this is equivalent to including a contingent link (Z, u, l, X). □

Computing a range constraint for an input parameter X of a process now requires the following: first we check whether P is dynamically controllable, if X is a non-contingent time point. If this check fails, then the service is not dynamically controllable, independent of the actual parameter value. If it is dynamically controllable, we compute u and l as described above.

Technically, we first add a non-contingent edge from Z to X labeled with a PLF over variable δ, and then solve δ, determining the lowest admissible upper-bound u for X. Then we add a non-contingent edge from X to Z labeled with a PLF over δ and solve it, determining the greatest admissible lower-bound l.

6.6.2 Computing Ranges for Input Parameters

Theorem 6.6.1 leads to the definition of a procedure for deriving range constraints for all input parameters of a process P, such that P is dynamically controllable. We define the procedure in Alg. 6.2.

Algorithm 6.2 Compute input parameter ranges

1: **Input:** P
2: $S := mapToCSTNU(P)$
3: $R := \emptyset$
4: **for** $X \in P.V^I$ **do**
5: $u := scon(S, X, Z)$
6: $l := -scon(S, Z, X)$
7: **if** $l < u$ **then**
8: **return** $\emptyset$
9: **end if**
10: $S := S \cup \{(Z, u, l, X)\}$
11: $R := R \cup \{(X, u, l)\}$
12: **end for**
13: **return** R

The procedure first maps P into an equivalent CSTNU S by applying the rules of Def. 5.3.2. Then the procedure iterates over all input parameters of P. At each iteration, it computes the broadest range constraint for a parameter, if possible. The range is computed by applying *scon* to derive the range bounds u and l, and verifying that $u < l$. If $u < l$, a corresponding contingent link for the parameter is added to the CSTNU. If at any iteration the computed values for u and l for a parameter X are such that $u \geq l$, then no valid range can be found for X which yields dynamic controllability guarantees. In such a case, the procedure returns the empty set, since no set of range restrictions for the parameters could be found.

6.6.3 Implementation and Evaluation

For checking the feasibility and scalability of the procedure, we implemented and performed experiments on Alg. 6.2. Complexity results for the operations in [81], however, are not yet available at the time of writing this work. Our experiments suggest that the approach is feasible for design-time support for processes of limited size.

We developed an algorithm in Java, which maps process models for service implementations compliant with Def. 4.3.1 to equivalent CSTNUs by applying the rules shown in Def. 5.3.2. Additionally, we implemented Algorithm 6.2 for computing parameter ranges for a given temporal network.

We ran our experiments on a Windows 10 physical machine with i7 CPU and 16GB of memory. We generated a set of 90 random processes resulting in CSTNUs of varying size between 5 and 20 nodes, and with varying number of temporal constraints. We then computed a parameter range using the implementation of the algorithm.

The complexity of the complete approach is clearly dominated by the complexity of the constraint propagation algorithm underlying *scon*, since the mapping to CSTNUs is linear in the number of process elements and in our experiments took always less than 15ms. The time required for computing the parameter range increases exponentially with the size of the processes. We consider the measured times, however, quite acceptable for design-time calculations. Our experiments demonstrate that the presented algorithms are feasible for practical applications, in which the process size is limited.

6.7 Comparison of the Two Approaches

We compare here the two approaches for computing restrictions for input temporal parameters proposed in this chapter.

With the approach based on the P-rules for constraint propagation, process designers can derive restrictions in terms of range constraints and constraints between parameters. Since the P-rules are designed for STNUs, the approach is suitable for process definitions which are mapped into STNUs. As discussed in Chapter 5, mapping process definitions into STNUs may lead to stricter results for the dynamic controllability check. In addition, also the parameter ranges derived from such an approach may be stricter than what could be necessary at run time, due to the absence of conditional executions in STNUs. In contrast, the PLF-based approach is based on mapping process definitions into CSTNUs, which allow capturing the semantics of conditional executions. Thus, this approach may derive less restrictive ranges than the approach based on applying the P-rules to a STNU.

With the PLF-based approach, for processes with one parameter, or a set of independent parameters, the broadest range for a parameter is unique, and the procedure will return this range. However, if there are dependencies between parameters, i.e. if an instantiation of one parameter might reduce the admissible interval for others, there might be several sets of ranges, and the procedure may yield different results for different sequences for the iteration over the input parameters. It is even possible that

for a given order of input parameters, no set of restrictions can be found, although one exists. In contrast, the P-rules based approach derives restrictions for parameters, considering all parameters at once. Thus, there is no particular order in which the restrictions are computed and, consequently, the resulting restrictions are unique. So if there exists a set of restrictions for the input parameters which yields dynamic controllability guarantees, the approach based on P-rules is guaranteed to find it.

Thus, we regard the approach based on the application of the P-rules as more suitable, since it always finds a set of restrictions for the input parameters of a process, if it exists. The broader applicability of the approach based on P-rules is also supported by the better scaling behavior, compared to the approach based on PLF propagation.

6.8 Conclusion

Service providers and process designers must fulfil the obligation to meet all temporal constraints specified in Temporal Service Level Agreements, while having to cope with the uncertain durations of service calls and activities. Temporal parameters specifying Temporal SLAs enable the definition of safe restrictions on their values, such that it is possible to cope with uncertain durations, which are observed only at run time. On the other hand, too tight restrictions limit the scope of applicability of a Web service composition or a process, thus providers and designers would like to offer flexibility of choice for the Temporal SLAs.

In this chapter we applied constraint propagation techniques for computing restrictions for the temporal parameters of a process. We have thus shown a possible approach for deriving parameter restrictions, giving an answer to our Research Subquestion 3.1.

We introduced a sound and complete method, based on a set of new STNU constraint propagation rules, to derive the broadest possible restrictions on the values temporal parameters may take. These restrictions give guarantees of dynamic controllability while defining the least restrictive Temporal SLAs.

Additionally, we have introduced an alternative method, based on propagating constraints labeled with piecewise-linear functions in CSTNUs. This second method takes into account conditional executions, thus computing potentially less restrictive

constraints. However, this second method guarantees to always find a set of parameter restrictions, only if there is one input parameter, or a set of independent parameters.

Finally, we have compared the two approaches, concluding that the approach based on propagating constraints in STNUs using the P-rules is more suitable due to its broader applicability. With the introduction and comparison of the two alternative techniques for computing parameter restrictions, we have given an answer to our Research Sub-question 3.2.

Our third Research Sub-question around the support to process designers for modeling process models which feature temporal variables still remains open. In the next chapter we build on the approach based on the new propagation rules, to design a procedure for the distributed negotiation of restrictions on temporal parameters in cross-organizational processes.

Chapter 7

Negotiating Restrictions for Temporal Parameters

7.1 Introduction

In Chapter 4 we have emphasized the importance of temporal obligations in cross-organizational business processes. We have shown how temporal constraints for the execution of these processes can be formulated as relations between admissible time-points of message exchanges and/or permissible ranges for the temporal parameters. Such constraints (e.g. the time interval for the reaction to the receipt of a message) are descriptive for one party ("when can I expect an answer from the other party") and prescriptive for the other party ("when do I have to send an answer?"). These temporal constraints are part of the negotiation of the communication protocols (choreography), respectively of Temporal Service Level Agreements, which are part of the specification for cross-organizational processes.

For process designers it is, therefore, of paramount importance to verify at design time, whether at run time it is possible to avoid any violation of these temporal obligations in the contracts. Violations must be avoided, independent of uncertainties about when other parties are sending messages within their agreed time frames, and of the uncertainties on the actual values for parameters, when they are within agreed ranges. So, technically, this means that process designers need to check the dynamic controllability of their processes.

The computations for checking dynamic controllability are, however, also a necessary part of the negotiation step for the design of cross-organizational processes. In particular, it is necessary for the negotiators, to calculate the consequences of any temporal obligation, and to compute which uncertainties (e.g., duration intervals) in the constraints of other parties are acceptable. In a nutshell, there is a need to calculate the trade-offs between the various temporal constraints, to support the negotiation of temporal commitments.

In Chapter 5 we have shown how to check, at design time, whether a process or Web service composition with temporal parameters is dynamically controllable. The requirement was that the process model is fully specified, i.e. all restrictions on the temporal variables are known. However, it is not always the case that such restrictions are known and thus modeled. So in Chapter 6 we have shown how to compute, at design time, the admissible restrictions on temporal parameters for single processes and Web service compositions, which hold dynamic controllability guarantees. These restrictions are valid for the single process using the parameters, however there are settings, such as cross-organizational processes, in which more processes have restrictions on the same parameters.

For cross-organizational processes computing these restrictions is more complex, due to the distributed design and execution, and the potentially different requirements of each partner. Indeed, the restrictions on a temporal parameters computed by one partner, may have counter-restrictions in the process of a collaborating partner. It is not necessarily the case that these restrictions coming from different sources are the same, or even compatible. So there is the need to reach agreements on the admissible restrictions on temporal parameters, such that the expectations on the parameter values of one (receiving) partner, correspond to obligations to provide the same parameter values for the other (sending) partner.

In this chapter we ask the question, how to achieve contracts between the parties of a cross-organizational business process with adequate restrictions on temporal parameters, admitting schedules free from time failures. In particular, we motivate the need for a negotiation protocol, and provide a sound and complete procedure for the negotiation of parameter restrictions in a cross-organizational process.

The contents of this chapter are based on the publications in [67], [68].

7.2 Cross-Organizational Process Model

Cross-organizational processes emerge from local processes interacting by exchanging messages according to some choreography. Here we do not focus on the definition of process views and the alignment of process views with choreographies or global processes, but we assume that a choreography or message exchange protocol has been defined already. Instead, we focus on the definition of temporal constraints for the process model. Temporal constraints may depend on each other, and there may be trade-offs between different temporal constraints.

We consider a cross-organizational process from the viewpoint of one party: a process model in which some of the steps are used for receiving and sending messages. Temporal constraints from such a viewpoint are local constraints and choreography constraints. Choreography constraints can only be formulated using the elements (events, temporal variables) which are known by both parties, i.e. elements of the process view. Local constraints can also include local events and local temporal variables.

We assume a global choreography for the cross-organizational process, with shared parameter space: this means that each local process has a view of the global process, and everybody knows all parameter constraints.

Temporal parameters are used in temporal constraints in different local processes. Thus, avoiding that these constraints are violated requires that the parameters take only values which are allowed by all processes using them, and are exchanged at agreed times as well. However, also the time in which a parameter exchange occurs can be modeled through parameters. So the exchange of temporal parameters requires a number of restrictions on the temporal parameters values, on which all the interested parties agree.

Now, if a set of parameter restrictions is given, as we have shown in Chapter 5, each local process can verify, whether these restrictions allow for temporally correct executions. However, often not all restrictions are known, and they must be computed. In particular, they must be computed by the interested parties (as we have shown in Chapter 6), and these parties must ensure that they agree on the results of the computation. Thus, a meta-protocol for negotiating adequate parameter restrictions between communicating processes is required.

By negotiation we mean finding an agreement on the choreography constraints, such that the cross-organizational process can be executed without time failures in any interacting local process, for all foreseeable situations.

7.3 Temporal Commitments

Parties in a cross-organizational process have to agree on temporal commitments, i.e. on constraints defined on admissible timepoints of communication events (sending and receiving messages) and on the ranges and restrictions for the temporal parameters. The goal is to agree on such a set of constraints, so that time failures can be avoided for all possible process instances, i.e. the processes are dynamically controllable.

For a receiving process, a range of an input parameter is a guarantee, that the values of this parameter will be within this range. On the other hand, it is an obligation for the sending process to send a value within the agreed range. If the range of an input variable is wider, then the uncertainty for the receiving process is higher and it is more difficult to avoid time failures. However, if the range of an output parameter is smaller, it is more difficult to send a valid parameter. Every parameter in a cross-organizational process is both output for some process and input for some other process.

Moreover, the ranges of the input parameters influence the possible ranges of output parameters, and dynamic controllability of the processes also depends on these ranges. Therefore, these ranges have to be negotiated between the different parties. Additionally, there might be inter-dependencies between parameters, leading to trade-offs between the ranges, i.e. a broader range for one parameters may reduce the range of other parameters.

For these negotiations, each party has to understand the dependencies and trade-offs between different ranges and other constraints. Since these dependencies are distributed over all the participating processes, these dependencies and their consequences can only be calculated in a distributed procedure.

For the negotiations the following computations are necessary:

- which ranges of the input parameters can be accepted,
- which ranges of output parameters can be guaranteed,

Algorithm 7.1 Infer parameter restrictions

```
1: Input: STNU S
2: if (¬check_dc(S)) then
3:    return False;
4: else
5:    while ((R := isRuleApplicable(S)) ≠ null) do
6:       S := applyRule(R, S);
7:       if (¬check_dc(S)) then
8:          return False;
9:       end if
10:   end while
11: end if
12: R := projectParameterConstraints(S, P);
13: return R;
```

- which constraints have to hold between parameters.

For the input parameters, we aim at computing the widest possible ranges and for the output parameters the most narrow ranges. In addition, we aim to determine the constraints between parameters representing the above mentioned trade-offs. In analogy to the previous chapters, for checking dynamic controllability and for computing admissible restrictions for temporal parameters, we adopt the approach based on mapping process definitions into temporal constraint networks.

7.4 Negotiating Parameter Restrictions

The prerequisite for negotiating parameter constraints between different local processes, is that each process partner has computed which constraints on the parameters need to hold for the dynamic controllability of the respective local process.

In Alg. 7.1 we repeat the procedure to locally infer parameter restrictions, which we introduced in Chapter 6. The inference procedure iteratively applies the basic inference rules of Sect. 6.2 to a dynamically controllable STNU S until quiescence (i.e. no new STNU edge can be derived). We call the resulting STNU the closure of S.

The closure can now be analyzed to extract parameter restrictions. We are mainly interested in the derived links between a parameter and *zero* and between 2 parameters. For a node n, a non-contingent edge $(zero, n, w)$ means that n is allowed to take at most value w, for the STNU to be dynamically controllable. A non-contingent edge

$(n, zero, -v)$ means that n has to take a value greater or equal to v, for the STNU to be dynamically controllable. We call $[v, w]$ the range restriction for n.

Lemma 7.4.1. *Let S be the closure of the STNU for a process P. Let $p \in param(S)$ (p is a temporal parameter of P) such that $\forall q \in param(S)\ \nexists (p, q, \delta), (q, p, \delta') \in S$. Let $(zero, p, w)$, $(p, zero, -v)$ be edges in S. Then $[v, w]$ is the broadest range of values for p which is compatible with the dynamic controllability of S.*

Proof. Let us assume that S as above is dynamically controllable. It is easy to see, that fixing p (by introducing edges $(zero, p, \overline{p})$ and $(p, zero, -\overline{p})$) to any value $\overline{p}$ either smaller than v, or larger than w, would introduce a negative loop in the STNU, thus making it not dynamically controllable. Now let us assume that S is dc but there is a value $v \leq \overline{p} \leq w$ for p, such that $S' = S \cup \{(zero, p, \overline{p}), (p, zero, -\overline{p})\}$ is not dc, i.e. there is a negative cycle in S'. Then with the Decomposability theorem [45] we can conclude that the negative cycle was already in S - which is a contradiction to the assumption that S is dynamically controllable. □

STNU edges derived between any two nodes m and n, represent constraints restricting the values for the timestamps of these nodes with respect to each other, which as well need to be fulfilled, in order for the STNU to be dynamically controllable. For instance, a derived edge $(m, n, 20)$ would mean *the timestamp of n must be no more than 20 time units after the timestamp of m.*

Lemma 7.4.2. *Let S be the closure of the STNU for a process P. Let p and q be nodes in S for temporal parameters. Let (p, q, w) be an edge in S. Then $q \leq p + w$ is the broadest constraint between the timestamps of p and q which is compatible with the dynamic controllability of S.*

Proof. It is easy to see, that fixing q (by introducing edges $(zero, q, \overline{q})$ and $(q, zero, -\overline{q})$) to any value $\overline{q}$ violating $q \leq p + w$, would introduce a negative loop in S, thus making it not dynamically controllable.

Now suppose that q is fixed (by introducing edges as above) to any value $\overline{q}$ fulfilling $q \leq p + w$ and all other constraints on q, and that a negative cycle is in S. Then the negative cycle must be due to some other configuration of constraints, which is a contradiction since S is dynamically controllable. □

7.5 A Procedure for Negotiating Constraints

We have shown how a process designer can compute restrictions for temporal parameters which were previously unknown by inferring constraints. We have shown that the inference procedure is sound and complete.

A local process can now communicate the inferred restrictions to the senders, resp. receivers of the parameters, with the aim of negotiating a shared set of restrictions. These are the restrictions which lead to a dynamically controllable STNU for each local process. Thus, fulfilling these restrictions guarantees that the local processes can execute in coordination while allowing the avoidance of any time failures, i.e. the global process is dynamically controllable.

A difficulty for such a negotiation procedure are inter-dependencies between parameters. Consider, as an example, a local process LP_0, which receives two input parameters: parameter p_1 is received from a process LP_1, and parameter p_2 from a different process LP_2. Through Alg. 7.1, it is discovered that, for the dynamic controllability of LP_0, both p_1 and p_2 have to take values in the range $[15, 20]$; and p_2 must be at least equal to $p_1 - 1$, and at most equal to $p_1 + 1$. So it is not sufficient that LP_0 asks LP_1 and LP_2 to provide parameters with values in the ranges, since, e.g., $p_1 = 16$ and $p_2 = 20$ would be invalid instantiations.

In Alg. 7.2 we provide a general formulation of an algorithm for deriving a shared set of restrictions on parameters. The algorithm is executed at each local process in the cross-organizational process.

First, each local process derives ranges for its parameters by applying the procedure for range inference of Alg. 7.1. Then, all inferred restrictions for the input parameters are sent to the respective senders (line 6), as requisites on the acceptable values for these parameters. Restrictions on output parameters, computed at the receivers are then received. For these received restrictions, corresponding non-contingent edges are added to the STNU (line 9). These edges represent the commitment to assign values to these parameters, which fulfil the requests of the receivers.

The inference and transmission of new restrictions repeats until no change occurs at any local STNU. The procedure works under the assumption stated in Sect. 7.2 that all parameter restrictions are shared between all parties of the process. Under such an assumption, each local process can easily observe when a fix-point is reached or a contradiction is derived, thus termination of the distributed procedure is ensured.

Algorithm 7.2 Compute, distribute, and receive parameter restrictions

1: $S := map_to_STNU(P)$;
2: **repeat**
3: $S' := S$;
4: $S' := infer_restrictions(S)$;
5: **for** $(n \in N | n.type = receive)$ **do**
6: $transmit_restrictions(n)$;
7: **end for**
8: **for** $(n \in N | n.type = send)$ **do**
9: $S' := S' \cup receive_restrictions(n)$;
10: **end for**
11: **until** $S' = S$;
12: **for** $(n \in N | n.type = receive)$ **do**
13: $make_contingent(S, n)$;
14: **end for**

Nevertheless, Alg. 7.2 does not require that internal constraints of local processes are exposed.

7.5.1 Interpretation of the Results

If in the procedures described any STNU reaches a negative cycle, then the cross-organizational process is not dynamically controllable, i.e. time failures may occur at run-time.

Otherwise, the algorithms Alg 7.1 and Alg 7.2 result in a set of dynamically controllable STNUs which are coherent in their constraints on the shared temporal parameters. The restrictions on these parameters can now be interpreted as follows: the constraints on input parameters define for which constellation of parameter values the network is dynamically controllable and are thus requirements sending processes have to fulfill. The restrictions on output parameters are guarantees which the receiving processes are allowed to assume. Constraints between parameters show the trade-offs to be resolved by further negotiations.

7.5.2 Soundness and Completeness of the Procedure

We observe that the soundness of the procedure shown in Alg. 7.2 is based (1) on the soundness of the procedure for inferring parameter restrictions called in line 4,

and (2) on the fact that introducing contingent edges specifying the ranges for input parameters in line 13 does not violate dynamic controllability.

For (1), it is sufficient to see that the inference procedure derives ranges which are not less restrictive than the ones derived by the more general rules of [17]. Thus, if a closure of the STNU applying Alg. 7.1 can be computed, the STNU is dynamically controllable.

For (2), we give proof to the following Lemma:

Lemma 7.5.1. *Let P be a process; let S be the STNU for P. Let $p_1, ..., p_n$ be input parameters, and $r_i = [p_{i_{min}}, p_{i_{max}}]$ be range restrictions for each p_i derived from the closure of S. Then $S \bigcup_i (zero, p_{i_{min}}, p_{i_{max}}, p_i)$ is dynamically controllable.*

Proof. For parameters for which only range restrictions are derived in the closure, the proof directly follows from Lemma 7.4.1. For parameters having additional restrictions, the proof follows from Lemma 7.4.2, and the requirement that all derived restrictions are enforced by the senders, in whose STNU these restrictions are non-contingent edges, i.e. they can be controlled. □

For the completeness of the procedure, it is sufficient to observe that, if a set of restrictions for the parameters exists, it is derived by a monotonic tightening of the constraints. On the other hand, if the tightening of constraints reaches a negative cycle, it necessarily means that the process is not dynamically controllable.

At the basis of Alg. 7.2 is the repeated application of the procedure for inferring restrictions. From previous results [114], the complexity of STNU constraint propagation algorithms is polynomial in the number of STNU nodes $O(N^4)$.

Existing figures [51] for the local application of such procedures to STNUs derived from process definitions indicate the practical applicability of the approach at design time. The local execution of Alg. 7.2 may invoke several times the inference procedure, if re-computation is necessary. However, each new invocation only restricts previously computed bounds, and the procedure stops in case a negative cycle is found. Thus, the overall complexity of the distributed execution of Alg. 7.2 at each local participant, is given by $O(N^4)$, with N the number of nodes in the global STNU.

7.6 Alternative Negotiation Approaches

There are several variants for implementing this basic procedure with different effects and assumptions, such as:

1. Global choreography for the cross-organizational process, with shared parameter space: *"everybody knows all parameter constraints"*;
2. Single central coordinator, with full access to all parameter ranges: the coordinator can act as a mediator between local processes and in particular recognizes when a fix-point is reached;
3. Only local views of a global unstructured process: each process designer has to take action to directly communicate computed requirements;
4. Global hierarchical process with no global view or central coordinator: a partial order of parameter and process dependencies can be constructed.
5. Fully distributed procedure based on distributed cycle detection (e.g. [13, 118]) for addressing dependencies between parameters.

These variants depend on the different approaches to the modeling of the cross-organizational process. The significantly different variants of the negotiation procedure, lead to consequent specific problems to address, such as communication of discovered restrictions to be enforced by parties who do not directly communicate, or identification of the distributed termination.

Addressing these problems is, however, out of the scope of this work. Nevertheless, they foster potential future research on the definition of meta-protocols to support the design of cross-organizational processes with temporal constraints expressed through parameters.

7.7 Conclusion

Service Level Agreements including temporal obligations are crucial for the definition of business collaborations. However, the need for secrecy has frequently precluded the formulation of cross-organizational temporal constraints. The introduction of temporal variables, in particular their use as temporal parameters, opens interesting new ways to define cross-organizational processes with Temporal SLAs. With temporal

parameters, process parties might exchange temporal information as data without jeopardizing the secrecy of process models.

In this chapter we have shown that temporal parameters are a highly adequate means for expressing temporal obligations and guarantees between the participants of a cross-organizational business process. However, we have pointed out the need for an agreement between the parties on the values these temporal parameters may take, since they need to ensure that each process can execute free from time failures. Thus, we have introduced the notion of negotiation of restrictions for temporal parameters.

Negotiating constraints on temporal parameters has to respect the need for keeping internals of the participating processes secret, while arriving at a solution which allows for the distributed control of processes in a way that no temporal constraint is violated. We have introduced procedures, based on the local computation of restrictions on the temporal parameters, which realize a negotiation of these restrictions between the interested parties.

The procedures proposed in this chapter successfully support the negotiation of temporal commitments through the computation of maximum ranges for input parameters and minimum ranges for output parameters in an effective way.

With the procedures define in this chapter we have therefore answered our Research Sub-question 3.3. Together with the contribution of Chapter 6, thus, we conclude that we were able to fully answer our third Research Question, around a comprehensive support to the design of process models featuring temporal variables.

Chapter 8

Related Work on Time Management in Business Processes

8.1 Introduction

Time management is a crucial facet of Business Processes Management. Appropriately managing time in business processes results in increased effectiveness of operation, reduced costs, and higher competitiveness. Poor time management, on the other hand, may results in sub-par operational results, potential compensation costs, and reduced trust. Thus, it should not be surprising that considerable research efforts have been conducted over the years around the various facets of time management in business processes. In particular, practitioners and researchers devoted intensive efforts around modeling, managing, and verifying the temporal qualities of process definitions.

In this chapter we first introduce research papers which provide an overview of time management in business processes from a research perspective, outline the challenges posed by time management, and analyze the practice of designing time-constrained processes. We then discuss the most relevant contributions, in form of a chronological evolution of research in time management, which led to the contribution of this thesis. Finally, we present other work on time management, which is still highly relevant, but did not have a significant influence on the contribution of this thesis.

8.2 Time Management in Business Processes: Overview and Challenges

A general overview of related work in time management for workflows and Business Process Management is given in [23]. In [58] as well, the authors give an overview of the most important works in the area, and discuss some of the most relevant challenges which time management poses. In [96], real world processes are analyzed to understand, and subsequently categorize, how time management, and in particular the modeling of temporal constraints, is put into practice. Here we briefly describe the content of these works, and position our contribution in the analysis of each of them.

8.2.1 The Temporal Perspective in Business Process Modeling: A Survey and Research Challenges [23]

In [23], the authors investigate existing methods for the specification of temporal constraints, classifying them into three different research areas: intra-organizational processes, Web service compositions, and cross-organizational processes. The comparison between the various approaches analyzed leads to the definition of three categories of temporal constraints, namely *(1) Intra-activity temporal constraints*, *(2) Inter-activity and Inter-event temporal constraints*, and *(3) Inter-process temporal constraints / Collaborative temporal constraints*. The first category comprises constraints which correspond to the duration constraints of the process models we introduced in Chapter 4, and constraints over cardinalities (i.e. limiting the repeated execution of a same activity within a time limit). To the second category belong constraints corresponding to upper- and lower-bound constraints between process events in the process models of Chapter 4. Finally, in the third category fall temporal constraints across local processes, which we have shown can be realized by the exchange of temporal parameters, and temporal constraints related to resources.

How does our work position itself, in the analysis of the specification of temporal constraints of [23]? Here we introduced a quite powerful process model, which enables the expression of 7 out of the 9 types of temporal constraints identified by the authors of [23]. The only 2 missing temporal constraints we do not allow designers to express are: (1) temporal constraints over cardinalities, since we exclude the definition of loops, and (2) constraints relating to resources, since resources are not included in the process

models of Chapter 4. In contrast, the analyzed approaches capable of expressing the most temporal constraints types, stand at 4 out of 9.

Additionally, [23] highlights further characteristics of the analyzed approaches: how the temporal requirements are modeled, which temporal properties are checked, and how the check is performed. Most of the analyzed approaches rely on established formalisms such as BPMN, Petri nets, Workflow Nets, or activity graphs, extended with temporal capabilities. Here we propose a conceptual workflow model which is quite generic yet expressive, thus allowing mappings of process definitions from established formalisms into our process model. These mappings are, however, out of the scope of this work. Most of the analyzed approaches verify the presence of deadlocks or the temporal consistency of process definitions, following model checking or algorithmic approaches. In contrast, here we do not verify whether a process definition may lead to deadlocks, but verify the more refined quality of dynamic controllability, by making use of established algorithms designed for Temporal Constraint Networks.

Finally, to summarize our contribution in terms of advantages and disadvantages as done in [23], we identify the major advantage of this work in the extent of expressible temporal constraints, and in the support to verify the dynamic controllability of process definitions. The disadvantage, instead, lies in the absence of support, for process designers, in making dynamically controllable process definitions which are verified to be not dynamically controllable.

8.2.2 Workflow Time Management Revisited [58]

The work in [58] summarizes the most important research efforts at the time of its writing. It considers researches which focus on the following aspects: modeling of temporal aspects in workflows, analysis of temporal properties, computation of schedules, minimization of time failures, monitoring, and modeling and checking temporal qualities in distributed workflows and Web service compositions.

Particular attention is devoted to stressing the importance of temporal constraints, in particular duration of activities, process deadlines, and lower- and upper-bound constraints. With our contribution, we propose a process model (and variants of it) which captures all the above temporal constraints.

Our contribution, in particular with the process models defined in Chapter 4 and the procedures defined in Chapters 5, 6, and 7, falls in the general strands of research

identified by [58] as efforts on *modeling of temporal aspects and analysis of workflows*, on *distributed workflows*, and on *QoS for Web service compositions*.

8.2.3 Process Time Patterns: A Formal Foundation [96]

The work in [96] (which gives formal backing to the work in [97]) provided a crucial consolidation in the understanding of which are the temporal requirements, practitioners typically set on their processes. The work is based on the rigorous analysis of an extensive number of real world processes (from various domains, such as healthcare, administration, industry, and finance), and on the knowledge of the authors in the field. Thus, stemming from both empirical evidence from the practice, and academic research, it constitutes a fundamental and highly reliable contribution.

The main contribution is the establishment of precise formal semantics for temporal patterns of constraints in processes, which are independent from any specific modeling language. With such precise formal semantics, all the actors involved in the lifecycle of a process are provided an established basis for common understanding. Further, formal semantics facilitate the design time and run time verification of process temporal qualities, and ease the detection of design flaws and temporal inconsistencies.

The result of the work is a set of 10 formally defined time patterns, which are partitioned into 4 categories:

- *Category 1: Durations and Time lags*, including patterns *TP1 - Time lags between activities*, *TP2 - Durations*, and *TP3 - Time lags between events*.
- *Category 2: Restricting execution times*, including patterns *TP4 - Fixed date elements*, *TP5 - Schedule restricted elements*, *TP6 - Time-based restrictions*, and *TP7 - Validity period*.
- *Category 3: Variability*, including pattern *TP8 - Time-dependent variability*.
- *Category 4: Recurrent process elements*, including patterns *TP9 - Cyclic elements*, and *TP10 - Periodicity*.

Which of the patterns formalized in [96] are supported by the process models presented in Chapter 4? With the constructs of duration and lower- and upper-bound constraints, possibly in combination, and using temporal variables for encoding events, it is possible to capture all time patters in *Category 1*, as well as patterns *TP4* and *TP7*.

On the contrary, due to the absence of more refined control flow constructs for iterated or time-dependent executions, it is not possible to model patterns *TP5*, *TP6*, and patterns in *Category 3* and *Category 4*.

With our work we provided process models which capture 5 out of the 10 time patterns established in [96]. We consider the extension of such process models with additional control flow constructs, to ultimately support the missing time patterns, as one of the research challenges, which our work opens up for further investigation.

8.3 Modeling, Management, and Verification of Temporal Aspects: A Historical Overview

Time management for business processes, in the sense of modeling and verification of temporal aspects, has its origins in the late 1990s. In this section we discuss the most relevant contributions towards the modeling of time constrained processes, and the verification of their temporal qualities, since the inception of the need for process time management. We focus in particular on previous research efforts, which have influenced this work the most, presenting them in rough chronological order of appearance. We discuss these approaches, identifying the extent to which general modeling of processes is addressed, the particular contribution from the time management perspective, and how they relate to our contribution.

8.3.1 C. Bussler (1998), and L. J. Zhao and E. A. Stohr (1999)

Initial contribution to the development of techniques for managing time in workflow systems are found in [14] and [153]. These early researches aim at the run time computation of schedules for workflow instances, with techniques originating from project management and timed graphs. Since these are very early contributions, they rely on previous work which is not specifically targeting workflow modeling and management. Thus, either they limit the set of considered control flow constructs (such as in [153], where parallel constructs are not allowed), or the proposed procedures do not scale well ([14] requires numerous recalculations). Uncertainties considered in these approaches are limited to the unknown decisions, which are taken at run time influencing the control flow, and do not include contingent durations. Additionally, since the main goal is the run time optimization of instance execution, time management

at design time is not supported. Nevertheless, these initial works contribute in formalizing requirements for workflow systems with time management capabilities, and provide initial implementations for such systems, which were not established before.

8.3.2 O. Marjanovic et al. (1999-2000)

A seminal contribution towards time management in business processes, which goes beyond scheduling issues, is found in [109], together with [108, 110]. The work originates from the observation that, at the time, the requirements on workflow systems exceeded their actual capabilities, in particular concerning time modeling and management. A number of such requirements is pointed out by the authors, such as detection of temporal inconsistencies, monitoring capabilities, and support to pro-active measures to avoid time failures. Thus, building on the explicitation of these needs, the work first introduces a framework which brings time modeling capabilities to the definition of production workflows. The framework is designed to empower process designers with methods to identify, classify, and model temporal constraints. Enabled temporal constraints in such a framework are durations, lower- and upper-bound constraints, and deadlines. In addition, a notion for temporal consistency is provided, as well as algorithms to verify temporal consistency of workflows at design time. These algorithms are built upon the shortest path partitioning algorithm, and on the Critical Path Method (CPM).

8.3.3 J. Eder et al. (1999-2000)

Among the earliest work on the modeling of temporal aspects is [57], followed by [54]. These papers propose a representation for workflows based on Workflow graphs, extended with constructs for the definition of activity durations, lower- and upper-bound constraints, and fixed-date constraints. In addition, towards the support to time management and verification of temporal qualities, [57] also introduces earliest and latest finishing times for activities. The resulting model is called *timed activity graph*. The paper provides also a number of algorithms for time management at build time, instantiation time, as well as run time. These algorithms in particular aim at defining schedules which avoid missing temporal constraints. The subsequent work in [54] refines the process model, by introducing an explicit distinction between conditional and unconditional workflow execution paths. The framework introduced in [57] is

consequently adapted with the additional representation of a workflow as an *unfolded workflow graph*, and the algorithms extended to cope with such a model.

8.3.4 E. Kafeza and K. Karlapalem (2000)

The work of [87] is based on Allen's interval algebra [3]. Interval algebra is used to specify temporal constraints in workflows, in the form of relations between workflow activities. In addition to temporal constraints, the temporal aspect of a workflow definition includes activity durations and deadlines. The authors propose a method to schedule temporally correct workflow definitions. Control flow constructs for workflow definitions include sequence, parallel splits, and conditional splits modeled as inclusive-or. As temporal correctness, the authors consider the capability of having an instance which meets all temporal constraints. Hence, the problem of finding a temporally correct schedule is modeled as a constraint satisfaction problem. Both design time and run time calculations are supported.

8.3.5 H. Zhuge et al. (2000-2001)

The work of Zhuge et al. at the turn of the century focuses on Internet-based workflows. In particular [156] and [155] are among the earliest works which concentrate on time management in geographically distributed processes. In these processes, it is possible that different time zones coexist, and need to be coordinated. Basically, the proposed workflow model considers the same control flow constructs as in [87]. The temporal aspects includes the design time definition of durations, and the corresponding run time observed durations. Thus, the authors stress the necessity for both design time and run time support for time management. They propose algorithms for computing schedules and checking temporal consistency, which explicitly take into account potential time differences resulting from the distributed nature of the processes.

8.3.6 C. Bettini et al. (2000-2002)

Scheduling in workflow systems whose activities are performed by different autonomous agents is investigated in [9] and its extension [10]. The goal is to minimize the times spent for communication and synchronization of the agents. At the basis of

the solution is the notion of *free schedule*, i.e. a schedule stating an interval for an agent to start the activity, but which gives the agents freedom of choice, when to terminate the activity, as long as the activity duration is within a specified interval. Hence, they consider activities to be contingent, as we do here. As for the control flow constructs, these as well are the same we consider in our work. The formalism used for computing schedules is the Simple Temporal Network of [45], thus without the extension of uncertain durations. Further, the instances which are made possible by the presence of conditional splits are treated separately by splitting the process model into the various traces, and performing computations in isolation. No merging of the results into a single process model is defined.

8.3.7 C. Combi and G. Pozzi (2003-2004), and C. Combi et al. (2007)

One of the most relevant papers in the area of conceptual modeling in relation to time, providing a comprehensive support to the modeling of temporal aspects in business processes, is [40]. It provides a first contribution in defining a conceptual model for workflows, which enables the specification of temporal constraints for workflow activities. The model is an extension of the conceptual model for workflows presented in [19], with added support for temporal constraints on single tasks (*task constraints*), based on schedules (*schedule-task constraints*), and between tasks (*inter-task constraints*). Further extensions with constraints on groups of tasks, and on free pairs of tasks, are also considered by the authors. The scope of the work in [40] is limited to a conceptual modeling discussion, and the paper does not describe a language for constraint specification. The subsequent work in [41], however, targets these points. It discusses different architectures for a workflow engine supporting time management, and proposes and describes a possible implementation. The implementation is based on an Oracle relational database, with the support of stored procedures for managing the temporal aspects. We regard these contributions as necessary early steps to put process time management in action, and to to consolidating the state of the art from the modeling perspective.

Subsequent work by Combi with other authors specifically focuses on the conceptual modeling of clinical workflows with temporal constraints in [31]. The clinical domain poses interesting research challenges to the modeling of temporal aspects: the authors propose a process model which includes the definition of durations, instantaneous events, explicit and implicit temporal constraints, periodicity, and delays. Thus,

the contribution of the paper is twofold: first, it gives clear motivation for the need to model a number of different temporal requirements in workflows. Then, it proposes a general conceptual model, which extends the previous work, by allowing the definition of a broader set of temporal aspects. The process model captures all the requirements from the clinical domain, but it is general enough to be suitable for other domains as well.

8.3.8 R. Lu et al. (2006)

Run time support to time management using temporal constraint networks is presented in [103]. The work proposes a framework for the modeling of business processes which takes into account the need for execution flexibility at run time. At the basis of the framework is the notion of process constraints, which define the various perspectives of a process definition. These perspectives are grouped into three classes of constraints: selection constraints, scheduling constraints, and resource constraints. The identified constraint classes may apply to the process level, and to the activity level. Basically, selection and schedule constraints correspond to the elements of the process model we introduced in Chapter 4. Resource constraints define which are the resources needed for executing the activities, and are not covered by our work. For schedule constraints, [103] relies on temporal constraint networks and Allen's interval algebra. For checking temporal consistency, the work proposes to use a path consistency algorithm. Checking temporal consistency is supported at design time and at run time as well; in addition, the work proposes a run time approach to raise exceptions in the case temporal constraint satisfaction cannot be guaranteed.

8.3.9 J. Eder and A. Tahamtan et al. (2007-2012)

Cross-organizational processes are the object of investigation in the work of Eder and Tahamtan. The authors, in conjunction with other authors in some of their papers, considers cross-organizational processes as choreographies implemented as federations of processes exposing views [55, 134, 135]. In [56] they verify the conformance of federated choreographies. Temporal conformance is addressed by the work in [60], while in [59] they take a probabilistic approach to choreography time management. The process model adopted in these research works is basically a timed workflow graph, for which specific algorithms are designed to derive schedules.

8.3.10 C. Combi et al. (2009-2019)

Combi and Posenato treat the problem of checking controllability of conceptual workflows in [36], and refine the treatment of the problem by addressing different types of controllability in [37]. They introduce a new conceptual model for processes, which features a number of temporal aspects (activity durations, delays and temporal constraints between activities, and deadlines). While in [36] they consider sequential and parallel executions only, in [37] they introduce also conditional executions and treat them through the notion of *workflow path*.

The initial work is focused on checking (strong) process controllability. Instead, in [37] three different kinds of controllability are addressed: strong, history-dependent, and weak; an algorithm which checks at design time all these three kinds of controllability is proposed. Complexity results for the checking algorithm are discussed in [38], while an extensive discussion on a run time flexible support to managing time constrained processes is found in [32]. The algorithms for checking controllabilities proposed in these and part of their subsequent contributions are all essentially based on solving the Simple Temporal Problem through Temporal Constraint Networks, with the adequate refinements, which lead to the adoption of Simple Temporal Networks with Uncertainty (STNUs) as the formal apparatus for temporal reasoning.

Subsequent work by Combi and Posenato with other authors in [29, 30] targets the modeling of clinical workflows, and the verification of their temporal correctness in terms of the different types of controllability.

Building on the expertise and the numerous contributions around temporal constraint networks, Combi and Posenato recently propose a work which considers time in relation to additional perspectives of a workflow model in [39]. In particular the authors address the relation between time and resources and run time resource constraints. They introduce the Conditional Simple Temporal Networks with Uncertainty and Resources (CSTNURs). However, for checking dynamic controllability the proposed approach is not to adapt the techniques built for temporal constraint networks; instead, the solution is to map CSTNURs into equivalent timed game automata, in a fashion similar to the work done by Zavatteri for checking dynamic controllability of Conditional Simple Temporal Networks with Uncertainty and Decisions (CSTNUDs) in [152].

8.3.11 A. Lanz et al. (2013-2019)

The work of Lanz et al. in [93] approaches the problem of verifying at design time and at run time the controllability of process definitions by projecting their temporal aspect into a Conditional Simple Temporal Network with Uncertainty (CSTNUs). The process model considered in the work includes sequences, parallel and conditional executions, and loops. However, since loops are assumed to be bounded in the number of iterations, they are unfolded into a sequence of iterations. Thus, the process model assumed in [93] basically corresponds to the process model we consider in this work, without temporal variables.

Subsequent work by Lanz et al. introduces an extension of temporal constraint networks, defined as Simple Temporal Networks with Partially Shrinkable Uncertainty (STNPSUs) [94]. STNPSUs allow capturing the temporal behavior of process modules, whose uncertain duration is mitigated by a degree of flexibility which can be controlled. The authors propose algorithms for checking dynamic controllability of STNPSUs, which are an adaptation of the classic checking algorithms of Morris and Muscettola [116]. STNPSUs are shown to support time management in modularized processes in [95, 123].

8.3.12 J. Eder and M. Franceschetti et al. (2018-2020)

At the basis of this thesis are the papers by Eder and Franceschetti, also with other authors. Temporal variables to model process events and other events not bound to the process control flow are introduced in [51]. There, we present a local process model featuring temporal variables, showing examples for the enabled temporal constraint patterns, and how to verify dynamic controllability of such process models based on STNU mapping.

Extending the expressiveness of process model with conditional executions based on the temporal advancement of a process at run time is part of the work in [50]; in [65] we further extended the modeling capabilities with the introduction of the construct of temporal loop, showing how a temporal bound to the loop iterations allows for checking dynamic controllability of such processes. Based on Simple Temporal Networks with Uncertainty is the work in [52]. There we introduce a new type of activity duration, *semi-contingent*, and define the notion of semi-dynamic controllability as a specialization of dynamic controllability, to address the need to avoid undesirable or

unacceptable run time behaviors such as the sudden termination of activities, caused by the need to meet certain temporal constraints. However, these contributions do not feature the use of temporal variables in process definitions, as they focus on control flow elements of process models, and their effect on the run time behavior.

We introduce a process model for Web service compositions exchanging temporal parameters, and verify their dynamic controllability when all temporal constraints are specified, in [62]. We show how to compute missing restrictions on temporal parameters for Web service compositions in [63], and subsequently propose a refined strategy in [66]. In the work in [64] we propose a technique for the dynamic binding of concrete services, aimed at extending the time allowed for a service call based on the run time observation of previous service calls, such that the Web service composition remains dynamically controllable.

A general discussion on the need for distributed procedures and meta-protocols for the negotiation of commonly agreed restrictions on temporal parameters in cross-organizational processes is pointed out in [67].

8.4 Other Work on Modeling and Reasoning about Time in Business Processes

A number of other research contributions proposed alternative approaches to the modeling and reasoning about the temporal aspects of business processes. Here we discuss some of the most relevant contributions, and contrast them to this work.

8.4.1 Temporal Logic and Event Calculus

Several works rely on formalisms such as temporal logic [122] or event calculus [132] for the monitoring of temporal constraints in business processes and Web service compositions: examples for such works are [106, 107, 113]. These approach allow for predictive and reactive time management in the case of foreseen or observed temporal violations. However, no approach has been proposed to verify the dynamic controllability of process definitions so far. Additionally, these formalisms are more suitable for the verification of certain temporal qualities of temporal constraint constellations, but do not provide adequate support for deriving additional temporal knowledge, as Temporal Constraint Networks do, as we have shown in the previous chapters.

8.4.2 Dynamic Condition Response Graphs

Dynamic Condition Response (DCR) Graphs [76] have been proposed as a declarative language for the expression of workflows. They are intended to be a bridge between the increased expressive power of temporal logic and the understandability of other languages for modeling workflows such as Petri nets. DCR Graphs *per se* allow modeling temporal dependencies between the various control flow elements of a workflow. The formalism allows for the verification of static properties such as safety and liveness. However, in their original formulation, DCR Graphs do not cover time from a quantitative perspective.

In [77] the authors introduce Timed DCR Graphs, an extension for DCR Graphs allowing for discrete time deadlines. The extended formalism supports the representation of cross-organizational processes, for which properties of safety and liveness can be checked. Checking of safety and liveness for such graphs is based on mappings to finite state automata. The work is subsequently extended with run time support for checking these properties, for proactive detection of violations, and various refinements in [6, 44].

Dynamic Condition Response Graph as a formalism have the benefit of being easily understandable and usable by process practitioners, as the work described above points out. Thus, they ease the checking for some qualities of process models. However, their applicability for time management is limited to these aforementioned properties, and they do not support the inference of additional temporal knowledge, such as prescriptive constraints whose satisfaction is a necessary condition for dynamic controllability, as we have shown in Chapter 6 and Chapter 7.

8.4.3 Modeling Durations and Temporal Constraints in BPMN

Stemming from the need to model time-constrained processes from the medical domain is the work in [34, 35, 42], where the authors address the problem of using BPMN for the expression of duration constraints and other temporal constraints at different abstraction levels. They present verification approaches to enable the management of duration violations in [34], and to verify the behavioral characteristics of process models based on timed automata in [42].

8.4.4 Process Mining

Process mining [139, 141] is a quite recent discipline in the broader field of Business Process Management, which is based on the analysis of existing traces of process runs. It is usually presented as the bridge between data science and business process management [140]. At the basis of process mining is the existence of large process log collections, which can be analyzed by adapting big data techniques. Traditionally, process mining is employed for three process management tasks: process discovery, process conformance checking, and process improvement.

Process mining techniques can be in particular adopted for predictive monitoring of the behavior of business processes at run time, as done in [105]. Predictive monitoring aims at identifying in advance possible violation of temporal constraints or Temporal SLAs. An overview of existing work around the application of process mining for the prediction of the run time behavior of process instances can be found in [111].

Process mining represents a significant and and worth to explore branch of research to provide support to the time management capabilities of processes. It is expected that research in the field will continue to grow in the future, addressing aspects such as the improvement of process models. Of course process improvement includes also the modeling of the temporal aspects. The requirement for the application of process mining techniques, however, is the existence of an extensive corpus of traces to analyze. The contribution of our work addresses the design of new processes, for which obviously there are no traces available. Thus, our work is orthogonal to the one on process mining.

8.4.5 Compliance Checking

An alternative approach to verify the (temporal) correctness of a process behavior is compliance checking [72]. Towards the checking of compliance of business processes are works such as [70, 104, 112, 137]. For checking business process compliance, it is required to have a set of business rules, to which a business process is required to adhere. The process definition, and its traces, may be compared against the rules specification, to verify whether the process conforms with the requirements expressed by these business rules.

Techniques for checking the temporal compliance of processes, thus, give process designers a way to verify whether their processes conform with certain temporal requirements. However, such temporal-oriented checks are rather limited with respect to the temporal correctness notions. Additionally, compliance checking merely targets the verification, but does not provide process designers tools for deriving additional, implicit, temporal knowledge.

8.4.6 Time in Service Level Agreements

A lot of research has been done on the formulation and verification of Service Level Agreements (SLAs) for business processes and Web service compositions. SLAs formalize promised and expected qualities by the the parties of a collaboration, and are particularly relevant in the context of services, in which they are typically advertised as part of a service specification. The need to support the expression of temporal requirements as part of the Service Level Agreements has been well argued in [117]. A number of works explored the issues of representing the (Temporal) SLAs and verifying whether they can be obeyed in Web service compositions, such as [46, 88, 89]. We regard the use of temporal variables, in conjunction with the specification of SLAs, as a new approach which can increase both the expressiveness of temporal constraints, and the secrecy of local process and service implementations.

Chapter 9

Conclusions

9.1 Conclusion

Time management is a fundamental aspect of Business Process Management. Time management involves a large number of tasks, such as defining temporal properties and restrictions which need to be obeyed during the execution of a process, verifying whether schedules meeting all temporal requirements are possible, computing such schedules, and possibly re-computing these schedules in reaction to run time observations.

With this work we have demonstrated that adequate time management must start right from the process design phase. A proper management of time from the early phases of the process lifecycle has several benefits. These benefits include reduced number of iterations of process design, and a significantly reduced risk of time failures during process operation. In the case of cross-organizational collaborations, this phase may result in the formalization of contracts such as Service Level Agreements, in which the various parties establish expectations and corresponding obligations regarding the temporal aspect. A number of choices need to be taken at these early stages of the process lifecycle, in order to carry on the time management tasks.

The first choice revolves around the question about which is the most suitable language for modeling a time constrained business process. An appropriate modeling language should allow capturing all the temporal information, which is relevant for the process to be modeled. This temporal information should include the set of process events, and should be expressive enough to model a variety of temporal constraints.

The modeling language should also allow combining such required expressiveness with additional requirements of process designers, such as the need to express temporal constraints, without harming the secrecy of the process implementation.

The choice of the modeling language is also influenced by the support it provides to checking the temporal correctness of process models. As we have discussed in this work, there is a plethora of different notions for temporal correctness, which one can choose to consider, depending on the application requirements. Thus, in addition to the choice of the modeling language, is also the choice on which notion for temporal correctness to strive for.

To enhance the modeling capabilities, with this work we have introduced a new modeling language in the form of a process model, which contemplates uncertainty of task durations, and features temporal variables for encoding event timestamps. We have shown that this language enables the expression of an extended set of temporal constraints. In particular, the use of temporal variables supports establishing temporal constraints, which do not necessarily refer to the executable events of a process. With temporal variables, it is possible to model events which occur outside the scope of a process, or to refer to timestamps occurring in artifacts. Setting temporal constraints on these temporal variables allows the expression of a number of temporal constraints, which previously existing languages did not permit.

We have reflected on the relevance of dynamic controllability as a refined notion for temporal correctness, motivating its suitability in the context of time-constrained business processes, since it acknowledges and takes into account the uncertain nature of durations. Thus, we have elected dynamic controllability as our target quality of a process definition.

For checking the dynamic controllability of processes designed using our proposed language, we proposed to rely on the existing procedures, which have been designed for Temporal Constraint Networks (TCNs). In particular, we motivated the Simple Temporal Network with Uncertainty (STNU), and the Conditional Simple Temporal Network with Uncertainty (CSTNU), as two TCN types which are expressive enough to carry the semantics of our process model. Thus, we proposed mapping rules for projecting the temporal aspect of a process definition onto a temporally equivalent STNU, resp. CSTNU. We have shown how to check the dynamic controllability of a process by checking it for its equivalent TCN, adopting established constraint propagation techniques.

To support process designers in building processes which are dynamically controllable, we developed and compared two techniques based on Temporal Constraint Networks. With these techniques, process designers can compute admissible restrictions for the temporal variables in a process. These restrictions limit the values temporal variables may take, to only those values which guarantee the dynamic controllability of the process they are used in. Thus, when these variables are instantiated during process execution, or they are passed to the process as parameters, with values which are compatible with the computed restrictions, it is ensured that it will be possible to dynamically schedule the process, fulfilling any other temporal constraint.

We additionally devoted our attention towards the support to the design of cross-organizational processes, in which the need to keep business secrecy intact is usually in contrast with the need to enable an effective collaboration, which requires sharing information between partners. In particular, we pointed out the potential risk of exposing process implementations, when temporal constraints across different processes need to be established and enforced. We have shown that with the exchange of temporal variables as parameters this risk vanishes, since they allow the transmission of event data without exposing any process internals. However, in order to reach dynamically controllable collaborations, the values of these parameters need to be restricted in a way that all processes accessing them can be scheduled in a timely manner. This requires finding an agreement on temporal parameter restrictions. We have thus proposed a distributed negotiation procedure, which computes, if they exist, common restrictions shared among the collaboration participants. These are the restrictions whose fulfillment guarantees that dynamic controllability of the distributed process can be achieved.

The temporal aspect of business processes has attracted the research effort of numerous experts. However, these research efforts did not result yet in a comprehensive support for the modeling and verification of the temporal aspect, which takes into account other requirements such as hierarchical process structuring, information hiding, or low coupling. With the introduction of temporal variables, as we have shown here, it is possible to mark a step forward in the support for a highly flexible design of time-constrained business processes with advanced guarantees of temporal correctness.

9.2 Answers to the Research Questions

We summarize here the answers we gave to the Research Questions and Research Sub-questions we posed in Chapter 3.

- **RQ1:** How can we support the modeling of the temporal aspects of business processes with data elements such as temporal variables?
  - **RSQ1.1:** *What are the elements constituting a process model which features temporal variables to model events as data elements?* We provided the formalization of three process model definitions (for intra-organizational processes, for Web service compositions, and for cross-organizational processes), which feature temporal variables to model events, and duration, upper-, and lower-bound constraints.
  - **RSQ1.2:** *Which is the enabled expressive power of a process model which features temporal variables to model events as data elements?* Through examples, we have shown patterns of temporal constraints which are enabled by the process models we introduced, and could not be expressed with existing process models.
  - **RSQ1.3:** *Which are the temporal semantics of a process model which features temporal variables to model events as data elements?* We provided the formalization of the definitions of scenario and valid scenario, which describe the temporal semantics of the proposed process models, by defining which process instances are valid.

  So, with the definition of the process models, the assessment of their expressive power, and the definition of their temporal semantics, we have shown how to support the modeling of the temporal aspects of business processes with temporal variables.

- **RQ2:** How can we check the temporal correctness for process models which feature temporal variables?
  - **RSQ2.1:** *Which is the most adequate notion for temporal correctness of process models with temporal variables?* We analyzed different existing notions of temporal correctness (satisfiability, strong, weak, and dynamic controllability), discussing their pros and cons. From this discussion, we motivated

dynamic controllability as the most suitable notion to consider for the temporal correctness of process models with temporal variables.

- **RSQ2.2:** *How to check the dynamic controllability of a process with temporal variables?* We provided the description and the implementation of a procedure, based on Temporal Constraint Networks, for verifying the dynamic controllability of processes with temporal variables. We have given proof of the correctness of the checking procedure. The implementation can be used in conjunction with existing tools for verifying the dynamic controllability of Temporal Constraint Networks.
- **RSQ2.3:** *Which Temporal Constraint Network types are the most suitable, for checking the dynamic controllability of processes which feature temporal variables?* We provided a comparison between two established Temporal Constraint Network types (STNU and CSTNU). We motivated the suitability of these Temporal Constraint Network types for representing the temporal aspect of processes with temporal variables.
- **RSQ2.4:** *How to map a process with temporal variables into a Temporal Constraint Network?* We provided the formalization of two sets of mapping rules, which transform processes with temporal variables into temporally equivalent STNUs or CSTNUs, respectively. We have given proof of the correctness of the resulting mappings.

So by defining the notion of temporal correctness, motivating STNUs and CSTNUs as adequate formalisms for temporal reasoning, defining mapping rules, and defining the checking procedure, we have shown how to check the temporal correctness of process models with temporal variables.

- **RQ3:** How can we support process designers in the design of temporally correct processes which feature temporal variables?
  - **RSQ3.1:** *How to compute, which are the allowed restrictions for the temporal variables used as input parameters of a process, such that when these variables assume values compatible with these restrictions, dynamic controllability is ensured?* We provided the description and the implementation of two procedures, both based on mapping processes into Temporal Constraint Networks, for computing restrictions for temporal parameters of a process. One procedure is based on new constraint propagation rules; the other procedure is based on an operation for computing constraint bounds. We compared the two

procedures, and motivated the benefits of the procedure based on the new constraint propagation rules over the alternative procedure.

- **RSQ3.2:** *Which constraint propagation techniques need to be applied, for computing restrictions for temporal variables, which ensure dynamic controllability of a process using these variables?* We provided formal definitions for new constraint propagation rules, to be used in one of the two procedures for computing temporal parameter restrictions. We gave proof of the correctness of these new constraint propagation rules. We have also introduced an operation for computing constraint bounds, which is at the basis of the alternative procedure for computing temporal parameter restrictions. We have given proof of the correctness of the application of this operation.
- **RSQ3.3:** *How to support the negotiation of computed restrictions on temporal variables, when these temporal variables are shared among several communicating processes, each having different requirements on a same variable?* We provided the description in natural language and in pseudo-code of a distributed procedure for negotiating temporal parameter restrictions in cross-organizational processes. The procedure is based on the procedure for computing temporal parameter restrictions, which we formalized and implemented.

So with the definition of constraint propagation rules, the definition of an operation to compute constraint bounds, the definition and implementation of procedures for computing restrictions, and for negotiating restrictions in cross-organizational settings, we have shown how to support process designers in the design of processes with temporal variables which are temporally correct.

Thus, we conclude that we have given answers to all the Research Questions and Sub-questions presented in Chapter 1, in terms of adequate artifacts.

9.3 Outlook

The contribution of this work invites further research in the area of time management for business processes. We discuss here some of the possibilities that the introduction of our process model with temporal variables opens up.

- *Further increased modeling expressiveness:* With the developments in the area of Temporal Constraint Networks, which add expressive power to the networks we

consider here, it is possible to expand the semantics of temporal constraints. As we have discussed in the previous chapters, CSTNUDs and STNPSUs are examples of recent refinements of TCNs, which bring increased expressiveness. With our work we have shown how to leverage on TCNs for verifying the temporal correctness and deriving temporal knowledge from a process definition. More expressive temporal networks enable the definition of more expressive process models, to better support process designers in modeling complex requirements of business processes. Examples of increased expressiveness at process level, which build on the advancements in the area of TCNs, are process models featuring time-based decisions, enabling the definition of temporal splits and temporal loops. These time-based decisions can be used to steer the control flow at run time, for instance by executing certain activities only if the elapsed time since process start is below a given threshold.

Since temporal variables are data objects, they are inherently connected with the process data flow. In cross-organizational processes, it may be required to optimize the data exchanges between the various partners. To this end, existing work addressing the possible transmissions of generic data can be found in [90, 92]. However, for temporal variables, the problem is more complex, since their values may be assigned at run time at unknown points in time, and they may be used to enforce temporal constraints. At the receiving side, they need to be known with a certain advance before they may actually be used, in order to properly schedule the process. Thus, optimization of temporal variables exchange is a challenging task, which requires particular consideration of scheduling. We regard the optimization of temporal variables exchange in cross-organizational processes as another potential future research direction.

- *Process mining for improved time management:* Process mining is a well-established discipline in the broader area of Business Process Management. While it may seem that Process mining for process discovery is a concluded research chapter, we consider mining of the temporal qualities of process definitions still open. Process mining may be used to analyze traces of already executed processes, and discover most frequent task durations, and most frequent values for temporal variables. These values may be used to restructure process definitions, for instance by reducing the intervals for certain task durations which are observed to take shorter times, and increasing the allowed ranges for temporal parameters which are expected as input, if these permit dynamic controllability.

Furthermore, it is expected that, in the future, process mining techniques will be increasingly employed to foster the improvement of existing process models. Examples are the replacement of sub-processes, or process fragments, which are discovered to be bottlenecks during the execution of a process, or improved allocation of resources for carrying out tasks, which are found out to be onerous. In the light of this, process mining techniques may be explored to improve the design of time-constrained processes. By analyzing process logs, for instance, it could be possible to discover stochastic distributions for durations. As a consequence, this would allow associating probability distributions for contingent duration intervals, adding new temporal knowledge to process models, which Temporal Constraint Networks cannot capture. Similarly, probability distributions may be derived also for temporal variables, possibly in association with the discovery of correlations with other data or process metrics.

Augmenting the activity duration intervals and the temporal variable ranges of our proposed process model with probability distributions may foster research into defining stochastic controllability notions, which are dependent on probabilistic knowledge. We regard this more complex temporal knowledge as a research challenge, whose solution may be beneficial for the design and scheduling of more realistic business process models.

- *Holistic modeling of time-constrained processes:* Process designers have to consider all the relevant aspects of a business process, and translate them into an executable model. These aspects do not only include the temporal aspect, but also the organizational, operational, and so on. For a faithful modeling, all these aspects need to be considered in correlation with each other.

  In this work we focused our attention on the sole temporal aspect. However, the temporal aspect in isolation reflects only partially the complex reality to be modeled. As a motivation for the introduction of temporal variables, among the others, we brought the need to refer to events which are not bound to the control flow of a process. The timestamps of these events may depend on factors such as the availability of resources, or of some artifact: for instance, the earliest time indicated in a contract for the completion of a project may be closer in time, if more human resources are assigned to the project. In turn, the number of human resources for working on the project may depend on the available budget. So there is the need to consider time not in isolation, but as an interdependent component of a process definition.

We consider the investigation into the inter-dependencies between the different aspects, including the temporal aspect, of a process model, as a research direction which could bring benefits for fostering improved design of time-constrained processes. These benefits include, as examples, finding the most adequate trade-offs between resource assignment and timeliness, reduced costs, better support to run time decisions, more efficient schedule computation.

- *Distributed execution of time-constrained business processes based on smart contracts:* In recent years we have witnessed the advent of blockchain technology [154]. With blockchain technology, and in particular the possibility of executing code on the blockchain in the form of smart contracts, new possibilities have emerged for realizing collaborations in an untrusted environment [24]. A new wave of research around the design and execution of business processes has gained momentum, pivoted around smart contracts. Several applications of smart contracts for the design and enactment of business processes have been proposed, such as Caterpillar [102]. In this context, the public yet trust-less nature of the blockchain poses challenges around the well known conflict between the need to expose information and the need to keep the level of business secrecy high [91]. Additionally, in spite of blockchain applications being in the spotlight, and the numerous research works around the support of blockchain technology to BPM, blockchain-based process design and execution is still at its infancy. In particular, time management is still a neglected aspect. As we have demonstrated here, temporal variables provide a new means to represent process temporal information, and enable the expression of temporal constraints in distributed settings, which preserves business secrecy. Thus, the introduction of temporal variables in process models, and the possibility to verify temporal correctness of processes with temporal variables, represent a basis for fostering research in secrecy-preserving time management of blockchain-based processes.

We regard the introduction of temporal variables to support the modeling of time in business processes as a valuable contribution, since it increases the expressiveness of process models, and fosters secrecy-preserving, time-constrained business collaboration. However, as we have briefly discussed here, we also consider our contribution as an enabling factor for much needed further research in the area of time management for business processes.

Bibliography

[1] Gustav Aagesen and John Krogstie. BPMN 2.0 for modeling business processes. In vom Brocke and Rosemann [146], pages 219–250. 19

[2] Shyan Akmal, Savana Ammons, Hemeng Li, and James C. Boerkoel Jr. Quantifying degrees of controllability in temporal networks with uncertainty. In Benton et al. [8], pages 22–30. 3

[3] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983. 113

[4] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. 28

[5] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using BPMN-Q and temporal logic. In Dumas et al. [48], pages 326–341. 21

[6] David A. Basin, Søren Debois, and Thomas T. Hildebrandt. In the nick of time: Proactive prevention of obligation violations. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 120–134. IEEE Computer Society, 2016. 119

[7] Anne Baumgraß, Nico Herzberg, Andreas Meyer, and Mathias Weske. BPMN extension for business process monitoring. In Fernand Feltz, Bela Mutschler, and Benoît Otjacques, editors, *Enterprise modelling and information systems architectures - EMISA 2014, Luxembourg, September 25-26, 2014*, volume P-234 of *LNI*, pages 85–98. GI, 2014. 20

[8] J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors. *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*. AAAI Press, 2019. 133, 143

[9] Claudio Bettini, Xiaoyang Sean Wang, and Sushil Jajodia. Free schedules for free agents in workflow systems. In *Seventh International Workshop on Temporal Representation and Reasoning, TIME 2000, Nova Scotia, Canada, July 7-9, 2000*, pages 31–37. IEEE Computer Society, 2000. 113

[10] Claudio Bettini, Xiaoyang Sean Wang, and Sushil Jajodia. Temporal reasoning in workflow systems. *Distributed Parallel Databases*, 11(3):269–306, 2002. 113

[11] Nikhil Bhargava and Brian C. Williams. Complexity bounds for the controllability of temporal networks with conditions, disjunctions, and uncertainty. *Artif. Intell.*, 271:1–17, 2019. 26

[12] László Böszörményi, Robert Eisner, and Herbert Groiss. Adding distribution to a workflow management system. In *10th International Workshop on Database & Expert Systems Applications, Florence, Italy, September 1-3, 1999, Proceedings*, pages 17–21. IEEE Computer Society, 1999. 37

[13] Lubos Brim, Ivana Cerná, and Lukás Hejtmánek. Distributed negative cycle detection algorithms. In Gerhard R. Joubert, Wolfgang E. Nagel, Frans J. Peters, and Wolfgang V. Walter, editors, *Parallel Computing: Software Technology, Algorithms, Architectures and Applications, PARCO 2003, Dresden, Germany*, volume 13 of *Advances in Parallel Computing*, pages 297–304. Elsevier, 2003. 104

[14] Christoph Bussler. Workflow instance scheduling with project management tools. In Roland R. Wagner, editor, *Ninth International Workshop on Database and Expert Systems Applications, Vienna, Austria, August 24-28, 1998, Proceedings*, pages 753–758. IEEE Computer Society, 1998. 111

[15] Massimo Cairo, Carlo Combi, Carlo Comin, Luke Hunsberger, Roberto Posenato, Romeo Rizzi, and Matteo Zavatteri. Incorporating decision nodes into conditional simple temporal networks. In Schewe et al. [130], pages 9:1–9:18. 26

[16] Massimo Cairo, Luke Hunsberger, Roberto Posenato, and Romeo Rizzi. A streamlined model of conditional simple temporal networks - semantics and equivalence results. In Schewe et al. [130], pages 10:1–10:19. 74

[17] Massimo Cairo and Romeo Rizzi. Dynamic controllability of simple temporal networks with uncertainty: Simple rules and fast real-time execution. *Theor. Comput. Sci.*, 797:2–16, 2019. xiii, 29, 31, 32, 80, 81, 87, 103

[18] Diego Calvanese, Silvio Ghilardi, Alessandro Gianola, Marco Montali, and Andrey Rivkin. Formal modeling and smt-based parameterized verification of data-aware BPMN. In Thomas T. Hildebrandt, Boudewijn F. van Dongen, Maximilian Röglinger, and Jan Mendling, editors, *Business Process Management - 17th International Conference, BPM 2019, Vienna, Austria, September 1-6, 2019, Proceedings*, volume 11675 of *Lecture Notes in Computer Science*, pages 157–175. Springer, 2019. 21

[19] Fabio Casati, Stefano Ceri, Barbara Pernici, and Giuseppe Pozzi. Conceptual modeling of workflows. In Papazoglou et al. [120], pages 281–306. 114

[20] Issam Chebbi, Schahram Dustdar, and Samir Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data Knowl. Eng.*, 56(2):139–173, 2006. 55

[21] Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. Toward a time-centric modeling of business processes in BPMN 2.0. In Edgar R. Weippl, Maria Indrawan-Santiago, Matthias Steinbauer, Gabriele Kotsis, and Ismail Khalil, editors, *The 15th International Conference on Information Integration and Web-based Applications & Services, IIWAS '13, Vienna, Austria, December 2-4, 2013*, page 154. ACM, 2013. 20

[22] Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. On enabling time-aware consistency of collaborative cross-organisational business processes. In Xavier Franch, Aditya K. Ghose, Grace A. Lewis, and Sami Bhiri, editors, *Service-Oriented Computing - 12th International Conference, ICSOC 2014, Paris, France, November 3-6, 2014. Proceedings*, volume 8831 of *Lecture Notes in Computer Science*, pages 351–358. Springer, 2014. 37

[23] Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. The temporal perspective in business process modeling: a survey and research challenges. *Service Oriented Computing and Applications*, 9(1):75–85, 2015. xi, 108, 109

[24] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016. 131

[25] Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation.

In Amedeo Cesta, Carlo Combi, and François Laroussinie, editors, *21st International Symposium on Temporal Representation and Reasoning, TIME 2014, Verona, Italy, September 8-10, 2014*, pages 27–36. IEEE Computer Society, 2014. 26

[26] Alessandro Cimatti, Luke Hunsberger, Andrea Micheli, Roberto Posenato, and Marco Roveri. Dynamic controllability via timed game automata. *Acta Inf.*, 53(6-8):681–722, 2016. 28, 29

[27] Workflow Management Coalition. *Terminology & Glossary, Issue 3.0*. 1999. xiii, 12, 14

[28] Carlo Combi and Mauro Gambini. Flaws in the flow: The weakness of unstructured business process modeling languages dealing with data. In Robert Meersman, Tharam S. Dillon, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2009, Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal, November 1-6, 2009, Proceedings, Part I*, volume 5870 of *Lecture Notes in Computer Science*, pages 42–59. Springer, 2009. 43

[29] Carlo Combi, Mauro Gambini, Sara Migliorini, and Roberto Posenato. Modelling temporal, data-centric medical processes. In Gang Luo, Jiming Liu, and Christopher C. Yang, editors, *ACM International Health Informatics Symposium, IHI '12, Miami, FL, USA, January 28-30, 2012*, pages 141–150. ACM, 2012. 116

[30] Carlo Combi, Mauro Gambini, Sara Migliorini, and Roberto Posenato. Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways. *IEEE Trans. Syst. Man Cybern. Syst.*, 44(9):1182–1203, 2014. 116

[31] Carlo Combi, Matteo Gozzi, José M. Juárez, Barbara Oliboni, and Giuseppe Pozzi. Conceptual modeling of temporal clinical workflows. In *14th International Symposium on Temporal Representation and Reasoning (TIME 2007), 28-30 June 2007, Alicante, Spain*, pages 70–81. IEEE Computer Society, 2007. 114

[32] Carlo Combi, Matteo Gozzi, Roberto Posenato, and Giuseppe Pozzi. Conceptual modeling of flexible temporal workflows. *ACM Trans. Auton. Adapt. Syst.*, 7(2):19:1–19:29, 2012. 116

[33] Carlo Combi, Luke Hunsberger, and Roberto Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited. In Joaquim Filipe and Ana L. N. Fred, editors, *Agents and*

Artificial Intelligence - 5th International Conference, ICAART 2013, Barcelona, Spain, February 15-18, 2013. Revised Selected Papers, volume 449 of *Communications in Computer and Information Science*, pages 314–331. Springer, 2013. 29

[34] Carlo Combi, Barbara Oliboni, and Francesca Zerbato. Modeling and handling duration constraints in BPMN 2.0. In Seffah et al. [131], pages 727–734. 119

[35] Carlo Combi, Barbara Oliboni, and Francesca Zerbato. A modular approach to the specification and management of time duration constraints in BPMN. *Inf. Syst.*, 84:111–144, 2019. 119

[36] Carlo Combi and Roberto Posenato. Controllability in temporal conceptual workflow schemata. In Dayal et al. [43], pages 64–79. 4, 116

[37] Carlo Combi and Roberto Posenato. Towards temporal controllabilities for workflow schemata. In Nicolas Markey and Jef Wijsen, editors, *TIME 2010 - 17th International Symposium on Temporal Representation and Reasoning, Paris, France, 6-8 September 2010*, pages 129–136. IEEE Computer Society, 2010. 65, 66, 116

[38] Carlo Combi and Roberto Posenato. On the complexity of temporal controllabilities for workflow schemata. In Sascha Ossowski and Paola Lecca, editors, *Proceedings of the ACM Symposium on Applied Computing, SAC 2012, Riva, Trento, Italy, March 26-30, 2012*, pages 60–66. ACM, 2012. 116

[39] Carlo Combi, Roberto Posenato, Luca Viganò, and Matteo Zavatteri. Conditional simple temporal networks with uncertainty and resources. *J. Artif. Intell. Res.*, 64:931–985, 2019. 26, 116

[40] Carlo Combi and Giuseppe Pozzi. Temporal conceptual modelling of workflows. In Il-Yeol Song, Stephen W. Liddle, Tok Wang Ling, and Peter Scheuermann, editors, *Conceptual Modeling - ER 2003, 22nd International Conference on Conceptual Modeling, Chicago, IL, USA, October 13-16, 2003, Proceedings*, volume 2813 of *Lecture Notes in Computer Science*, pages 59–76. Springer, 2003. 114

[41] Carlo Combi and Giuseppe Pozzi. Architectures for a temporal workflow management system. In Hisham Haddad, Andrea Omicini, Roger L. Wainwright, and Lorie M. Liebrock, editors, *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), Nicosia, Cyprus, March 14-17, 2004*, pages 659–666. ACM, 2004. 114

[42] Carlo Combi, Pietro Sala, and Francesca Zerbato. Driving time-dependent paths in clinical BPMN processes. In Seffah et al. [131], pages 743–750. 119

[43] Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo A. Reijers, editors. *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings*, volume 5701 of *Lecture Notes in Computer Science*. Springer, 2009. 137, 142

[44] Søren Debois, Thomas T. Hildebrandt, and Tijs Slaats. Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes. In Nikolaj Bjørner and Frank S. de Boer, editors, *FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings*, volume 9109 of *Lecture Notes in Computer Science*, pages 143–160. Springer, 2015. 119

[45] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, 1991. 3, 22, 29, 100, 114

[46] Adela del-Río-Ortega, Antonio Manuel Gutiérrez, Amador Durán Toro, Manuel Resinas, and Antonio Ruiz Cortés. Modelling service level agreements for business process outsourcing services. In Jelena Zdravkovic, Marite Kirikova, and Paul Johannesson, editors, *Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings*, volume 9097 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 2015. 121

[47] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Inf. Softw. Technol.*, 50(12):1281–1294, 2008. 21

[48] Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors. *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*. Springer, 2008. 133, 148

[49] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A Reijers. Fundamentals of business process management. 2018. 11

[50] Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of orchestrations with temporal SLA: encoding temporal XOR in CSTNUD. In Maria Indrawan-Santiago, Eric Pardede, Ivan Luiz Salvadori, Matthias Steinbauer, Is-

mail Khalil, and Gabriele Anderst-Kotsis, editors, *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2018, Yogyakarta, Indonesia, November 19-21, 2018*, pages 234–242. ACM, 2018. 45, 117

[51] Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of business processes with temporal variables. In Chih-Cheng Hung and George A. Papadopoulos, editors, *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019*, pages 40–47. ACM, 2019. 3, 7, 43, 47, 65, 103, 117

[52] Johann Eder, Marco Franceschetti, and Josef Lubas. Scheduling processes without sudden termination. In Selmin Nurcan, Iris Reinhartz-Berger, Pnina Soffer, and Jelena Zdravkovic, editors, *Enterprise, Business-Process and Information Systems Modeling - 21st International Conference, BPMDS 2020, 25th International Conference, EMMSAD 2020, Held at CAiSE 2020, Grenoble, France, June 8-9, 2020, Proceedings*, volume 387 of *Lecture Notes in Business Information Processing*, pages 117–132. Springer, 2020. 117

[53] Johann Eder and Wolfgang Gruber. A meta model for structured workflows supporting workflow transformations. In Yannis Manolopoulos and Pavol Návrat, editors, *Advances in Databases and Information Systems, 6th East European Conference, ADBIS 2002, Bratislava, Slovakia, September 8-11, 2002, Proceedings*, volume 2435 of *Lecture Notes in Computer Science*, pages 326–339. Springer, 2002. xiii, 19

[54] Johann Eder, Wolfgang Gruber, and Euthimios Panagos. Temporal modeling of workflows with conditional execution paths. In Ibrahim et al. [83], pages 243–253. 112

[55] Johann Eder, Nico Kerschbaumer, Julius Köpke, Horst Pichler, and Amirreza Tahamtan. View-based interorganizational workflows. In Boris Rachev and Angel Smrikarov, editors, *Proceedings of the 12th International Conference on Computer Systems and Technologies, CompSysTech 2011, Vienna, Austria, June 16-17, 2011*, pages 1–10. ACM, 2011. 55, 115

[56] Johann Eder, Marek Lehmann, and Amirreza Tahamtan. Conformance test of federated choreographies. In Ricardo Jardim-Gonçalves, Jörg P. Müller, Kai Mertins, and Martin Zelm, editors, *Enterprise Interoperability II - New Challenges and Industrial Approaches, Proceedings of the 3th International Conference on Inter-*

operability for Enterprise Software and Applications, IESA 2007, March 27-30, 2007, Funchal, Madeira Island, Portugal, pages 223–234. Springer, 2007. 37, 115

[57] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Time constraints in workflow systems. In Jarke and Oberweis [84], pages 286–300. 112

[58] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Workflow time management revisited. In Jr. et al. [86], pages 207–213. xi, 1, 108, 109, 110

[59] Johann Eder, Horst Pichler, and Amirreza Tahamtan. Probabilistic time management of choreographies. In Danilo Ardagna, Massimo Mecella, and Jian Yang, editors, *Business Process Management Workshops, BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers*, volume 17 of *Lecture Notes in Business Information Processing*, pages 443–454. Springer, 2008. 115

[60] Johann Eder and Amirreza Tahamtan. Temporal consistency of view based interorganizational workflows. In Roland H. Kaschek, Christian Kop, Claudia Steinberger, and Günther Fliedl, editors, *Information Systems and e-Business Technologies, 2nd International United Information Systems Conference, UNISCON 2008, Klagenfurt, Austria, April 22-25, 2008, Proceedings*, volume 5 of *Lecture Notes in Business Information Processing*, pages 96–107. Springer, 2008. 3, 115

[61] Clarence A. Ellis and Gary J. Nutt. Modeling and enactment of workflow systems. In Marco Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993, 14th International Conference, Chicago, Illinois, USA, June 21-25, 1993, Proceedings*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1993. 18

[62] Marco Franceschetti and Johann Eder. Checking temporal service level agreements for web service compositions with temporal parameters. In Elisa Bertino, Carl K. Chang, Peter Chen, Ernesto Damiani, Michael Goul, and Katsunori Oyama, editors, *2019 IEEE International Conference on Web Services, ICWS 2019, Milan, Italy, July 8-13, 2019*, pages 443–445. IEEE, 2019. 7, 65, 118

[63] Marco Franceschetti and Johann Eder. Computing ranges for temporal parameters of composed web services. In *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services, iiWAS 2019, Munich, Germany, December 2-4, 2019*, pages 537–545. ACM, 2019. 7, 78, 118

[64] Marco Franceschetti and Johann Eder. Dynamic service binding for time-aware service compositions. In *23rd IEEE International Enterprise Distributed Object Com-*

puting Workshop, EDOC Workshops 2019, Paris, France, October 28-31, 2019, pages 146–151. IEEE, 2019. 7, 78, 118

[65] Marco Franceschetti and Johann Eder. Towards checking dynamic controllability of processes with temporal loops. In José Ignacio Panach, Renata S. S. Guizzardi, and Daniela Barreiro Claro, editors, *Proceedings of the ER Forum and Poster & Demos Session 2019 on Publishing Papers with CEUR-WS co-located with 38th International Conference on Conceptual Modeling (ER 2019), Salvador, Brazil, November 4, 2019*, volume 2469 of *CEUR Workshop Proceedings*, pages 1–14. CEUR-WS.org, 2019. 117

[66] Marco Franceschetti and Johann Eder. Computing admissible temporal slas for web service compositions. In *2020 IEEE International Conference on Web Services, ICWS 2020, Beijing, China, October 20-23, 2020 (forthcoming)*, 2020. 7, 43, 78, 118

[67] Marco Franceschetti and Johann Eder. Designing decentralized business processes with temporal constraints. In *CAiSE Forum 2020, Held at CAiSE 2020, Grenoble, France, June 8-9, 2020 (forthcoming)*, 2020. 8, 43, 96, 118

[68] Marco Franceschetti and Johann Eder. Negotiating temporal commitments in cross-organizational business processes. In *27th International Symposium on Temporal Representation and Reasoning, TIME 2020, Bolzano - Bozen, Italy, September 23-25, 2020 (forthcoming)*, 2020. 8, 43, 96

[69] Denis Gagné and André Trudel. Time-bpmn. In Birgit Hofreiter and Hannes Werthner, editors, *2009 IEEE Conference on Commerce and Enterprise Computing, CEC 2009, Vienna, Austria, July 20-23, 2009*, pages 361–367. IEEE Computer Society, 2009. 20

[70] Luciano García-Bañuelos, Nick van Beest, Marlon Dumas, Marcello La Rosa, and Willem Mertens. Complete and interpretable conformance checking of business processes. *IEEE Trans. Software Eng.*, 44(3):262–290, 2018. 120

[71] Nicolas Genon, Patrick Heymans, and Daniel Amyot. Analysing the cognitive effectiveness of the bpmn 2.0 visual notation. In Brian A. Malloy, Steffen Staab, and Mark van den Brand, editors, *Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers*, volume 6563 of *Lecture Notes in Computer Science*, pages 377–396. Springer, 2010. 20

[72] Guido Governatori, Zoran Milosevic, and Shazia Wasim Sadiq. Compliance checking between business processes and business contracts. In *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), 16-20 October 2006, Hong Kong, China*, pages 221–232. IEEE Computer Society, 2006. 120

[73] Paul W. P. J. Grefen and Yigal Hoffner. Crossflow-cross-organizational workflow support for virtual organizations. In *Proceedings of the Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises, Sydney, Australia, March 23-24, 1999*, pages 90–91. IEEE Computer Society, 1999. 55

[74] Michael Hammer. What is business process management? In vom Brocke and Rosemann [146], pages 3–16. 1

[75] Thomas S. Heinze, Wolfram Amme, and Simon Moser. A restructuring method for WS-BPEL business processes based on extended workflow graphs. In Dayal et al. [43], pages 211–228. 19

[76] Thomas T. Hildebrandt and Raghava Rao Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In Kohei Honda and Alan Mycroft, editors, *Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software, PLACES 2010, Paphos, Cyprus, 21st March 2010*, volume 69 of *EPTCS*, pages 59–73, 2010. 119

[77] Thomas T. Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats, and Francesco Zanitti. Contracts for cross-organizational workflows as timed dynamic condition response graphs. *J. Log. Algebraic Methods Program.*, 82(5-7):164–185, 2013. 119

[78] David Hollingsworth. Wfmc: Workflow reference model. Technical report, Workflow Management Coalition, 1995. xiii, 13, 15

[79] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007. 28

[80] Luke Hunsberger and Roberto Posenato. Sound-and-complete algorithms for checking the dynamic controllability of conditional simple temporal networks with uncertainty. In Natasha Alechina, Kjetil Nørvåg, and Wojciech Penczek, editors, *25th International Symposium on Temporal Representation and Reasoning,*

TIME 2018, Warsaw, Poland, October 15-17, 2018, volume 120 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 29

[81] Luke Hunsberger and Roberto Posenato. Propagating piecewise-linear weights in temporal networks. In Benton et al. [8], pages 223–231. 88, 89, 90, 91

[82] Luke Hunsberger, Roberto Posenato, and Carlo Combi. The dynamic controllability of conditional stns with uncertainty. *CoRR*, abs/1212.2005, 2012. 24, 25, 29, 66, 74

[83] Mohamed T. Ibrahim, Josef Küng, and Norman Revell, editors. *Database and Expert Systems Applications, 11th International Conference, DEXA 2000, London, UK, September 4-8, 2000, Proceedings*, volume 1873 of *Lecture Notes in Computer Science*. Springer, 2000. 139, 143

[84] Matthias Jarke and Andreas Oberweis, editors. *Advanced Information Systems Engineering, 11th International Conference CAiSE'99, Heidelberg, Germany, June 14-18, 1999, Proceedings*, volume 1626 of *Lecture Notes in Computer Science*. Springer, 1999. 140, 149

[85] Kurt Jensen. Coloured petri nets and the invariant-method. *Theor. Comput. Sci.*, 14:317–336, 1981. 18

[86] Janis A. Bubenko Jr., John Krogstie, Oscar Pastor, Barbara Pernici, Colette Rolland, and Arne Sølvberg, editors. *Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE*. Springer, 2013. 140, 152

[87] Eleanna Kafeza and Kamalakar Karlapalem. Gaining control over time in workflow management applications. In Ibrahim et al. [83], pages 232–242. 113

[88] Slim Kallel, Anis Charfi, Tom Dinkelaker, Mira Mezini, and Mohamed Jmaiel. Specifying and monitoring temporal properties in web services compositions. In Rik Eshuis, Paul W. P. J. Grefen, and George Angelos Papadopoulos, editors, *Seventh IEEE European Conference on Web Services (ECOWS 2009), 9-11 November 2009, Eindhoven, The Netherlands*, pages 148–157. IEEE Computer Society, 2009. 121

[89] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Representation, verification, and computation of timed properties in web. In *2006 IEEE International Conference on Web Services (ICWS 2006), 18-22 September 2006, Chicago, Illinois, USA*, pages 497–504. IEEE Computer Society, 2006. 121

[90] Julius Köpke, Marco Franceschetti, and Johann Eder. Analyzing data-flow implementations for distributed execution of inter-organizational processes. In Maria Indrawan-Santiago, Matthias Steinbauer, Ivan Luiz Salvadori, Ismail Khalil, and Gabriele Anderst-Kotsis, editors, *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2017, Salzburg, Austria, December 4-6, 2017*, pages 502–510. ACM, 2017. 129

[91] Julius Köpke, Marco Franceschetti, and Johann Eder. Balancing privity and enforceability of bpm-based smart contracts on blockchains. In Claudio Di Ciccio, Renata Gabryelczyk, Luciano García-Bañuelos, Tomislav Hernaus, Rick Hull, Mojca Indihar Stemberger, Andrea Ko, and Mark Staples, editors, *Business Process Management: Blockchain and Central and Eastern Europe Forum - BPM 2019 Blockchain and CEE Forum, Vienna, Austria, September 1-6, 2019, Proceedings*, volume 361 of *Lecture Notes in Business Information Processing*, pages 87–102. Springer, 2019. 131

[92] Julius Köpke, Marco Franceschetti, and Johann Eder. Optimizing data-flow implementations for inter-organizational processes. *Distributed and Parallel Databases*, 37(4):651–695, 2019. 55, 129

[93] Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Controllability of time-aware processes at run time. In Robert Meersman, Hervé Panetto, Tharam S. Dillon, Johann Eder, Zohra Bellahsene, Norbert Ritter, Pieter De Leenheer, and Dejing Dou, editors, *On the Move to Meaningful Internet Systems: OTM 2013 Conferences - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013, Graz, Austria, September 9-13, 2013. Proceedings*, volume 8185 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2013. 117

[94] Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Simple temporal networks with partially shrinkable uncertainty. In Stéphane Loiseau, Joaquim Filipe, Béatrice Duval, and H. Jaap van den Herik, editors, *ICAART 2015 - Proceedings of the International Conference on Agents and Artificial Intelligence, Volume 2, Lisbon, Portugal, 10-12 January, 2015*, pages 370–381. SciTePress, 2015. 26, 117

[95] Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Controlling time-awareness in modularized processes. In Rainer Schmidt, Wided Guédria, Ilia Bider, and Sérgio Guerreiro, editors, *Enterprise, Business-Process and Information Systems Modeling - 17th International Conference, BPMDS 2016, 21st*

International Conference, EMMSAD 2016, Held at CAiSE 2016, Ljubljana, Slovenia, June 13-14, 2016, Proceedings, volume 248 of *Lecture Notes in Business Information Processing*, pages 157–172. Springer, 2016. 117

[96] Andreas Lanz, Manfred Reichert, and Barbara Weber. Process time patterns: A formal foundation. *Inf. Syst.*, 57:38–68, 2016. xi, 48, 50, 60, 108, 110, 111

[97] Andreas Lanz, Barbara Weber, and Manfred Reichert. Time patterns for process-aware information systems. *Requir. Eng.*, 19(2):113–141, 2014. 110

[98] Charles E. Leiserson and James B. Saxe. A mixed-integer linear programming problem which is efficiently solvable. *J. Algorithms*, 9(1):114–128, 1988. 22

[99] Angel Lagares Lemos, Florian Daniel, and Boualem Benatallah. Web service composition: A survey of techniques and tools. *ACM Comput. Surv.*, 48(3):33:1–33:41, 2016. 52

[100] Yuh-Zen Liao and Chak-Kuen Wong. An algorithm to compact a VLSI symbolic layout with mixed constraints. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2(2):62–69, 1983. 22

[101] Dongsheng Liu, Jianmin Wang, Stephen C. F. Chan, Jiaguang Sun, and Li Zhang. Modeling workflow processes with colored petri nets. *Comput. Ind.*, 49(3):267–281, 2002. 18

[102] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, and Ingo Weber. Caterpillar: A blockchain-based business process management system. In Robert Clarisó, Henrik Leopold, Jan Mendling, Wil M. P. van der Aalst, Akhil Kumar, Brian T. Pentland, and Mathias Weske, editors, *Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling (BPM 2017), Barcelona, Spain, September 13, 2017*, volume 1920 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017. 131

[103] Ruopeng Lu, Shazia Wasim Sadiq, Vineet Padmanabhan, and Guido Governatori. Using a temporal constraint network for business process execution. In Gillian Dobbie and James Bailey, editors, *Database Technologies 2006, Proceedings of the 17th Australasian Database Conference, ADC 2006, Hobart, Tasmania, Australia, January 16-19 2006*, volume 49 of *CRPIT*, pages 157–166. Australian Computer Society, 2006. 115

[104] Linh Thao Ly, Stefanie Rinderle-Ma, David Knuplesch, and Peter Dadam. Monitoring business process compliance using compliance rule graphs. In Robert Meersman, Tharam S. Dillon, Pilar Herrero, Akhil Kumar, Manfred Reichert, Li Qing, Beng Chin Ooi, Ernesto Damiani, Douglas C. Schmidt, Jules White, Manfred Hauswirth, Pascal Hitzler, and Mukesh K. Mohania, editors, *On the Move to Meaningful Internet Systems: OTM 2011 - Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011, Hersonissos, Crete, Greece, October 17-21, 2011, Proceedings, Part I*, volume 7044 of *Lecture Notes in Computer Science*, pages 82–99. Springer, 2011. 120

[105] Fabrizio Maria Maggi, Chiara Di Francescomarino, Marlon Dumas, and Chiara Ghidini. Predictive monitoring of business processes. In Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff, editors, *Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings*, volume 8484 of *Lecture Notes in Computer Science*, pages 457–472. Springer, 2014. 120

[106] Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil M. P. van der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf, editors, *Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings*, volume 6896 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2011. 118

[107] Fabrizio Maria Maggi, Michael Westergaard, Marco Montali, and Wil M. P. van der Aalst. Runtime verification of ltl-based declarative process models. In Sarfraz Khurshid and Koushik Sen, editors, *Runtime Verification - Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers*, volume 7186 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2011. 118

[108] Olivera Marjanovic. Dynamic verification of temporal constraints in production workflows. In *Australasian Database Conference, ADC 2000, Canberra, Australia, January 31 - February 3, 2000*, pages 74–81. IEEE Computer Society, 2000. 112

[109] Olivera Marjanovic and Maria E. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowl. Inf. Syst.*, 1(2):157–192,

1999. 112

[110] Olivera Marjanovic and Maria E. Orlowska. Time management in dynamic workflows. In Yanchun Zhang, Marek Rusinkiewicz, and Yahiko Kambayashi, editors, *The Proceedings of the Second International Symposium on Cooperative Database Systems for Advanced Applications, CODAS 1999, Wollongong, Australia, March 27-28, 1999*, pages 138–149. Springer, 1999. 112

[111] Alfonso Eduardo Márquez-Chamorro, Manuel Resinas, and Antonio Ruiz-Cortés. Predictive monitoring of business processes: A survey. *IEEE Trans. Serv. Comput.*, 11(6):962–977, 2018. 120

[112] Giovanni Meroni, Luciano Baresi, Marco Montali, and Pierluigi Plebani. Multi-party business process compliance monitoring through iot-enabled artifacts. *Inf. Syst.*, 73:61–78, 2018. 120

[113] Marco Montali, Fabrizio Maria Maggi, Federico Chesani, Paola Mello, and Wil M. P. van der Aalst. Monitoring business constraints with the event calculus. *ACM Trans. Intell. Syst. Technol.*, 5(1):17:1–17:30, 2013. 118

[114] Paul Morris. A structural characterization of temporal dynamic controllability. In Frédéric Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*, volume 4204 of *Lecture Notes in Computer Science*, pages 375–389. Springer, 2006. 24, 103

[115] Paul H. Morris and Nicola Muscettola. Managing temporal uncertainty through waypoint controllability. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 1253–1258. Morgan Kaufmann, 1999. 3, 23, 29

[116] Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 1193–1198. AAAI Press / The MIT Press, 2005. xiv, 29, 70, 117

[117] Carlos Müller, Antonio Ruiz Cortés, and Pablo Fernandez. Temporal-awareness in slas: Why should we be concerned? In Elisabetta Di Nitto and Matei Ripeanu,

editors, *Service-Oriented Computing - ICSOC 2007 Workshops, International Workshops, Vienna, Austria, September 17, 2007, Revised Selected Papers*, volume 4907 of *Lecture Notes in Computer Science*, pages 165–173. Springer, 2007. 121

[118] Gabriele Oliva, Roberto Setola, Luigi Glielmo, and Christoforos N. Hadjicostis. Distributed cycle detection and removal. *IEEE Trans. Control. Netw. Syst.*, 5(1):194–204, 2018. 104

[119] Business Process Modeling Notation OMG. Version 1.0. *OMG Final Adopted Specification, Object Management Group*, 190, 2006. 19

[120] Mike P. Papazoglou, Stefano Spaccapietra, and Zahir Tari, editors. *Advances in Object-Oriented Data Modeling*. In Papazoglou et al. [120], 2000. 135, 148

[121] Carl Adam Petri. Kommunikation mit automaten. *PhD thesis, Bonn, Institut für Instrumentelle Mathematik*, 1962. 17

[122] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. 118

[123] Roberto Posenato, Andreas Lanz, Carlo Combi, and Manfred Reichert. Managing time-awareness in modularized processes. *Software and Systems Modeling*, 18(2):1135–1154, 2019. 117

[124] Ivo Raedts, Marija Petkovic, Yaroslav S. Usenko, Jan Martijn E. M. van der Werf, Jan Friso Groote, and Lou J. Somers. Transformation of BPMN models for behaviour analysis. In Juan Carlos Augusto, Joseph Barjis, and Ulrich Ultes-Nitsche, editors, *Modelling, Simulation, Verification and Validation of Enterprise Information Systems, Proceedings of the 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS-2007, In conjunction with ICEIS 2007, Funchal, Madeira, Portugal, June 2007*, pages 126–137. INSTICC PRESS, 2007. 21

[125] Hajo A. Reijers and Jan Mendling. Modularity in process models: Review and effects. In Dumas et al. [48], pages 20–35. 36

[126] Hajo A. Reijers, Jan Mendling, and Remco M. Dijkman. Human and automatic modularizations of process models to enhance their comprehension. *Inf. Syst.*, 36(5):881–897, 2011. 36

[127] Wasim Sadiq and Maria E. Orlowska. Applying graph reduction techniques for identifying structural conflicts in process models. In Jarke and Oberweis [84], pages 195–209. 19

[128] Wasim Sadiq and Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2):117–134, 2000. 19

[129] Khodakaram Salimifard and Mike Wright. Petri net-based modelling of workflow systems: An overview. *Eur. J. Oper. Res.*, 134(3):664–676, 2001. 18

[130] Sven Schewe, Thomas Schneider, and Jef Wijsen, editors. *24th International Symposium on Temporal Representation and Reasoning, TIME 2017, October 16-18, 2017, Mons, Belgium*, volume 90 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. 134

[131] Ahmed Seffah, Birgit Penzenstadler, Carina Alves, and Xin Peng, editors. *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*. ACM, 2017. 137, 138

[132] Murray Shanahan. The event calculus explained. In Wooldridge and Veloso [151], pages 409–430. 118

[133] Robert E. Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28(4):769–779, 1981. 22

[134] Amirreza Tahamtan and Johann Eder. View driven federation of choreographies. In Ngoc Thanh Nguyen, Radoslaw P. Katarzyniak, and Shyi-Ming Chen, editors, *Advances in Intelligent Information and Database Systems*, volume 283 of *Studies in Computational Intelligence*, pages 145–156. Springer, 2010. 115

[135] Amirreza Tahamtan and Johann Eder. View driven inter-organisational workflows. *Int. J. Intell. Inf. Database Syst.*, 6(2):93–112, 2012. 115

[136] Ioannis Tsamardinos, Thierry Vidal, and Martha E. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003. 3, 24

[137] Han van der Aa, Henrik Leopold, and Hajo A. Reijers. Checking process compliance on the basis of uncertain event-to-activity mappings. In Eric Dubois and Klaus Pohl, editors, *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, volume 10253 of *Lecture Notes in Computer Science*, pages 79–93. Springer, 2017. 120

[138] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998. 18

[139] Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011. 120

[140] Wil M. P. van der Aalst. Using process mining to bridge the gap between BI and BPM. *IEEE Computer*, 44(12):77–80, 2011. 120

[141] Wil M. P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, R. P. Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos C. A. M. Buijs, Andrea Burattin, Josep Carmona, Malú Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian W. Günther, Antonella Guzzo, Paul Harmon, Arthur H. M. ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maria Maggi, Donato Malerba, R. S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith D. Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Thandar Wynn. Process mining manifesto. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer, 2011. 120

[142] Wil M. P. van der Aalst, Alexander Hirnschall, and H. M. W. Verbeek. An alternative way to analyze workflow graphs. In Anne Banks Pidduck, John Mylopoulos, Carson C. Woo, and M. Tamer Özsu, editors, *Advanced Information Systems Engineering, 14th International Conference, CAiSE 2002, Toronto, Canada, May 27-31, 2002, Proceedings*, volume 2348 of *Lecture Notes in Computer Science*, pages 535–552. Springer, 2002. 19

[143] Wil M. P. van der Aalst and Mathias Weske. The P2P approach to interorganizational workflows. In Klaus R. Dittrich, Andreas Geppert, and Moira C. Norrie, editors, *Advanced Information Systems Engineering, 13th International Conference, CAiSE 2001, Interlaken, Switzerland, June 4-8, 2001, Proceedings*, volume 2068 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2001. 55

[144] Thierry Vidal and Hélène Fargier. Contingent durations in temporal csps: From consistency to controllabilities. In *4th International Workshop on Temporal Representation and Reasoning, TIME '97, Daytona Beach, Florida, USA, May 10-11, 1997*, pages 78–85. IEEE Computer Society, 1997. 23, 27, 28

[145] Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.*, 11(1):23–45, 1999. 23

[146] Jan vom Brocke and Michael Rosemann, editors. *Handbook on Business Process Management 1, Introduction, Methods, and Information Systems, 2nd Ed*, International Handbooks on Information Systems. Springer, 2015. 133, 142

[147] Kenji Watahiki, Fuyuki Ishikawa, and Kunihiko Hiraishi. Formal verification of business processes with temporal and resource constraints. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Anchorage, Alaska, USA, October 9-12, 2011*, pages 1173–1180. IEEE, 2011. 66

[148] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, 2nd Edition.* Springer, 2012. 1, 10

[149] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014. 5

[150] Petia Wohed, Wil M. P. van der Aalst, Marlon Dumas, Arthur H. M. ter Hofstede, and Nick Russell. On the suitability of BPMN for business process modelling. In Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors, *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*, pages 161–176. Springer, 2006. 19

[151] Michael J. Wooldridge and Manuela M. Veloso, editors. *Artificial Intelligence Today: Recent Trends and Developments*. Volume 1600 of Wooldridge and Veloso [151], 1999. 149, 151

[152] Matteo Zavatteri and Luca Viganò. Conditional simple temporal networks with uncertainty and decisions. *Theor. Comput. Sci.*, 797:77–101, 2019. 26, 29, 68, 116

[153] J. Leon Zhao and Edward A. Stohr. Temporal workflow management in a claim handling system. In *Proceedings of the international joint conference on Work activities coordination and collaboration 1999, San Francisco, California, USA, February 22-25, 1999*, pages 187–195. ACM, 1999. 111

[154] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In George Karypis and Jia Zhang, editors, *2017 IEEE International Congress on Big Data, BigData Congress 2017, Honolulu, HI, USA, June 25-30, 2017*, pages 557–564. IEEE Computer Society, 2017. 131

[155] Hai Zhuge, To-Yat Cheung, and Hung Keng Pung. A timed workflow process model. *J. Syst. Softw.*, 55(3):231–243, 2001. 113

[156] Hai Zhuge, Hung Keng Pung, and To-Yat Cheung. Timed workflow: Concept, model, and method. In Qing Li, Z. Meral Özsoyoglu, Roland R. Wagner, Yahiko Kambayashi, and Yanchun Zhang, editors, *WISE 2000, Proceedings of the First International Conference on Web Information Systems Engineering, Volume I (Main Program), Hong Kong, China, June 19-21, 2000*, pages 183–189. IEEE Computer Society, 2000. 113

[157] Wlodzimierz M. Zuberek. Timed petri nets and preliminary performance evaluation. In Jacques Lenfant, Barry R. Borgerson, Daniel E. Atkins, Keki B. Irani, David Kinniment, and Hideo Aiso, editors, *Proceedings of the 7th Annual Symposium on Computer Architecture, La Baule, France, May 6-8, 1980*, pages 88–96. ACM, 1980. 18

[158] Stefan Zugal, Jakob Pinggera, Barbara Weber, Jan Mendling, and Hajo A. Reijers. Assessing the impact of hierarchy on model understandability - A cognitive perspective. In Jörg Kienzle, editor, *Models in Software Engineering - Workshops and Symposia at MODELS 2011, Wellington, New Zealand, October 16-21, 2011, Reports and Revised Selected Papers*, volume 7167 of *Lecture Notes in Computer Science*, pages 123–133. Springer, 2011. 36

[159] Michael zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In Jr. et al. [86], pages 429–443. 20, 43