Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Adaptive Time- and Process-Aware Information Systems

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

Vorgelegt von:
Andreas Lanz
geboren in Friedrichshafen

2017

Nur Wissen überlebt, nämlich
indem es den denkenden
Menschen buchstäblich informiert,
den Zustand (state) seines
Gehirns, ändert.

(Joseph Weizenbaum, 1923-2008)

Amtierender Dekan:	Prof. Dr. Frank Kargl
Gutachter:	Prof. Dr. Manfred Reichert
	Prof. Dr. Roberto Posenato
	Prof. Dr. Akhil Kumar
Tag der Promotion:	07. Juni 2017

For Melanie. . .

Your love and patience with me have not gone unnoticed. Thanks for always being there when I needed you.

Abstract

For the digitized enterprise the proper handling of the temporal aspects of its business processes is vital. Delivery times, appointments and deadlines must be met, processing times and durations be monitored, and optimization objectives shall be pursued. However, contemporary Process-Aware Information Systems (PAISs)—the go-to solution for the computer-aided support of business processes—still lack a sophisticated support of the time perspective. Hence, there is a high demand for a more profound support of temporal aspects in PAISs. Accordingly, both the specification and the operational support of temporal aspects constitute fundamental challenges for the further development and dissemination of PAISs. The aim of this thesis is to propose a framework for supporting the time perspective of business processes in PAISs. As PAISs enable the design, execution and evolution of business processes, the designated framework must support these three fundamental phases of the process life cycle.

The ATAPIS framework proposed by this thesis essentially comprises three major components.

First, *a universal and comprehensive set of time patterns* is provided. Respective time patterns represent temporal concepts commonly found in business processes and are based on empirical evidence. In particular, they provide a universal and comprehensive set of notions for describing temporal aspects in business processes. Moreover, a precise formal semantics for each of the time patterns is provided based on an in-depth analysis of a large set of real-world use cases. Respective formal semantics enable the proper integration of the time patterns into PAISs. In turn, the latter will allow for the specification of time-aware process schemas.

Second, *a generic framework for implementing the time patterns based on their formal semantics* is developed. The framework and its techniques enable the verification of time-aware process schemas regarding their temporal consistency, i. e., their ability to be successfully executed without violating any of their temporal constraints. Subsequently, the framework is extended to consider advanced aspects like the contingent nature of activity durations and alternative execution paths as well. Moreover, an algorithm as well as techniques for executing and monitoring time-aware process instances in PAISs is provided. Based on the presented concepts, it becomes possible to ensure that a time-aware process instance may be executed without violating any of its temporal constraints.

Third, *a set of change operations for dynamically modifying time-aware process instances during run time* is suggested. Respective change operations ensure that a modified time-aware process instance remains temporally consistent after the respective modification. Moreover, to reduce the complexity involved when applying multiple change operations a sophisticated approximation-based technique is presented. Overall, the developed change operations allow providing the flexibility required by business processes in practice.

Altogether, the ATAPIS framework provides fundamental concepts, techniques and algorithms for integrating the time perspective into PAISs. As beauty of this framework the specification, execution and evolution of business processes is supported by an integrated approach.

Kurzfassung

Im Zeitalter der Digitalisierung ist für Unternehmen der richtige Umgang mit den zeitlichen Aspekten ihrer Geschäftsprozesse zweifellos von großer Wichtigkeit. Lieferzeiten, Termine und Fristen müssen eingehalten, Durchlaufzeiten und Dauern überwacht und wirtschaftliche Ziele innerhalb einer gewünschten Zeit verfolgt werden. Dennoch fehlt es modernen Prozessorientierten Informationssystemen (POIS) – der bevorzugten Lösung für computergestützte Geschäftsprozesse – bisher an einer umfassenden Berücksichtigung der Zeitperspektive. Daraus resultiert bei vielen Unternehmen eine hohe Nachfrage nach einer tiefgreifenden Unterstützung zeitlicher Aspekte durch POIS. Grundsätzlich stellen sowohl die Spezifikation als auch die operative Unterstützung zeitlicher Aspekte grundlegende Herausforderungen für die weitere Entwicklung und Verbreitung von POIS dar. Das Ziel dieser Dissertation ist es, ein geeignetes Rahmenwerk zur umfassenden Unterstützung der Zeitperspektive von Geschäftsprozessen in POIS zu entwickeln. Da POIS die Modellierung, Ausführung und Evolution von Geschäftsprozessen ermöglichen, soll das zu realisierende Rahmenwerk ebenfalls diese drei Phasen des Prozesslebenszyklus unterstützen.

Das im Rahmen dieser Dissertation vorgeschlagene ATAPIS-Framework besteht im Wesentlichen aus drei Komponenten:

Erstens wird eine *umfassenden Menge universeller „Time Pattern“(engl. Zeit Muster)* vorgestellt. Die Time Pattern beruhen auf empirischen Untersuchungen und repräsentieren zeitliche Konzepte, welche häufig in Geschäftsprozessen vorkommen. Insbesondere stellen sie eine universelle und umfassende Menge an Konzepten für die Beschreibung der zeitlichen Aspekte von Geschäftsprozessen dar. Darüber hinaus wird eine präzise formale Semantik für jedes der Time Pattern definiert, welche auf einer eingehenden Analyse einer großen Menge realer Anwendungsfälle basiert. Die formale Semantik ermöglicht insbesondere die tiefergehende Integration der Time Pattern in POIS und damit die Spezifikation von zeitbehafteten Prozessschemata.

Zweitens wird ein *generisches Rahmenwerk für die Umsetzung der Time Pattern auf der Grundlage ihrer formalen Semantik* entwickelt. Das Rahmenwerk und seine Techniken ermöglichen die Überprüfung zeitbehafteter Prozessschemata hinsichtlich ihrer zeitlichen Konsistenz, das heißt hinsichtlich der Möglichkeit hiervon abgleitete Prozessinstanzen erfolgreich und ohne Verletzung einer ihrer zeitlichen Bedingungen auszuführen. Anschließend wird das Rahmenwerk um erweiterte Aspekte, etwa die unkontrollierbare Natur der Dauern von Aktivitäten und die Berücksichtigung alternativer Ausführungspfade, erweitert. Darüber hinaus werden ein Algorithmus und entsprechende Techniken für die Ausführung und Überwachung zeitbehafteter Prozessinstanzen in POIS entwickelt. Basierend auf den vorgestellten Konzepten lässt sich sicherstellen, dass eine zeitbehaftete Prozessinstanz ohne Verletzung einer ihrer zeitlichen Bedingungen ausgeführt werden kann.

Drittens wird eine Reihe von *wohldefinierten Operationen für die dynamische Änderung zeitbehafteter Prozessinstanzen zur Laufzeit* bereitgestellt. Die Änderungsoperationen stellen sicher, dass eine modifizierte zeitbehaftete Prozessinstanz auch nach ihrer Änderung zeitlich konsistent bleibt. Zur Reduzierung der Komplexität bei der typischen Anwendung mehrerer Änderungsoperationen wird eine approximative Technik präsentiert. Die Änderungsoperationen erlauben es uns, die für Geschäftsprozesse in der Praxis erforderliche Flexibilität zu realisieren.

Insgesamt stellt das in dieser Arbeit vorgestellte ATAPIS-Framework grundlegende Konzepte, Technologien und Algorithmen für die Integration der Zeitperspektive in POIS zur Verfügung. Der Vorteil des vorgestellten Ansatzes besteht darin, dass die Spezifikation, Durchführung und Evolution von Geschäftsprozessen durch einen integrierten Ansatz unterstützt werden kann. Die Ergebnisse der Arbeit ermöglichen damit die tiefgreifende Integration der Zeitperspektive in heutige und zukünftige POIS.

List of Publications

This cumulative dissertation is a consolidated report of the research results obtained during the author's Ph. D. project. The detailed results have been published in the following refereed papers:

LWR14 A. Lanz, B. Weber, and M. Reichert. Time patterns for process-aware information systems. *Requirements Engineering*, 19(2):113–141, 2014. DOI: 10.1007/s00766-012-0162-3

LRW16 A. Lanz, M. Reichert, and B. Weber. Process time patterns: A formal foundation. *Information Systems*, 57:28–68, 2016. DOI: 10.1016/j.is.2015.10.002

LPCR13 A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controllability of time-aware processes at run time. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences — Proceedings of the 21st International Conference on Cooperative Information Systems (CoopIS'13)*, number 8185 in Lecture Notes in Computer Science, pages 39–56. Springer, September 2013. DOI: 10.1007/978-3-642-41030-7_4

LR14 A. Lanz and M. Reichert. Dealing with changes of time-aware processes. In *Business Process Management — Proceedings of the 12th International Conference on Business Process Management (BPM'14)*, volume 8659 of *Lecture Notes in Computer Science*, pages 217–233. Springer, September 2014. DOI: 10.1007/978-3-319-10172-9_14

LWR10 A. Lanz, B. Weber, and M. Reichert. Workflow time patterns for process-aware information systems. In *Enterprise, Business-Process and Information Systems Modeling — Proceedings of the 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010, held at CAiSE 2010*, volume 50 of *Lecture Notes in Business Information Processing*, pages 94–107. Springer, June 2010. DOI: 10.1007/978-3-642-13051-9_9

Additionally, parts of this thesis have been published in the following publications.

- B. Weber, A. Lanz, and M. Reichert. Time patterns for process-aware information systems: A pattern-based analysis. Technical report, University of Ulm, Germany, 2008. URL http://dbis.eprints.uni-ulm.de/527/
- A. Lanz, B. Weber, and M. Reichert. Time patterns in process-aware information systems - a pattern-based analysis - revised version. Technical Report UIB-2009-05, University of Ulm, Germany, 2009. URL http://dbis.eprints.uni-ulm.de/648/
- A. Lanz, M. Reichert, and P. Dadam. Making business process implementations flexible and robust: Error handling in the AristaFlow BPM Suite. In *Proceedings of the CAiSE'10 Forum - Information Systems Evolution*, volume 592 of *CEUR Workshop Proceedings*, pages 18–25. CEUR, 2010
- A. Lanz, U. Kreher, M. Reichert, and P. Dadam. Enabling process support for advanced applications with the AristaFlow BPM Suite. In *Proceedings of the Business Process Management 2010 Demonstration Track*, number 615 in CEUR Workshop Proceedings, September 2010
- A. Lanz, M. Reichert, and P. Dadam. Robust and flexible error handling in the AristaFlow BPM Suite. In *Proc. CAiSE'10 Forum, Information Systems Evolution*, number 72 in Lecture Notes in Business Information Processing, pages 174–189. Springer, June 2010. DOI: 10.1007/978-3-642-17722-4_13
- I. Barba, A. Lanz, B. Weber, M. Reichert, and C. del Valle. Optimized time management for declarative workflows. In *Proceedings of the 13th International Conference BPMDS 2012, 17th International Conference EMMSAD 2012, and 5th EuroSymposium*, volume 113 of *Lecture Notes in Business Information Processing*, pages 195–210. Springer, June 2012. DOI: 10.1007/978-3-642-31072-0_14
- A. Lanz, M. Reichert, and B. Weber. A formal semantics of time patterns for process-aware information systems. Technical Report UIB-2013-02, University of Ulm, 2013. URL http://dbis.eprints.uni-ulm.de/894/
- A. Lanz, J. Kolb, and M. Reichert. Enabling personalized process schedules with time-aware process views. In *Advanced Information Systems Engineering Workshops (CAiSE'13 Workshops)*, volume 148 of *Lecture Notes in Business Information Processing*, pages 205–216. Springer, 2013. DOI: 10.1007/978-3-642-38490-5_20
- A. Lanz and M. Reichert. Process change operations for time-aware processes. Technical Report UIB-2014-01, University of Ulm, 2014. URL http://dbis.eprints.uni-ulm.de/1027/
- A. Lanz and M. Reichert. Enabling time-aware process support with the ATAPIS toolset. In *Proceedings of the BPM Demo Sessions 2014*, volume 1295 of *CEUR Workshop Proceedings*, pages 41–45. CEUR, 2014

- A. Lanz, R. Posenato, C. Combi, and M. Reichert. Simple temporal networks with partially shrinkable uncertainty (extended version). Technical Report UIB-2014-05, Ulm University, 2014. URL http://dbis.eprints.uni-ulm.de/1124/
- A. Lanz, R. Posenato, C. Combi, and M. Reichert. Simple temporal networks with partially shrinkable uncertainty. In *Proceedings of the 7th International Conference on Agents and Artificial Intelligence (ICAART'15)*, pages 370–381. SCITEPRESS, January 2015. DOI: 10.5220/0005200903700381
- A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controlling time-awareness in modularized processes (extended version). Technical Report UIB-2015-01, University of Ulm, 2015. URL http://dbis.eprints.uni-ulm.de/1133/
- A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controlling time-awareness in modularized processes. In *Proceedings of the 17th International Conference BPMDS 2016*, pages 157–172. Springer, 2016. DOI: 10.1007/978-3-319-39429-9_11

Contents

Part I

Problem Description and Backgrounds

1 Introduction

In today's fast-paced business world, companies crave for an effective and efficient control over their *business processes* in order to stay competitive in their market [8, 93, 123]. As a consequence, IT support for analyzing, modeling, executing, monitoring, and evolving business processes is becoming increasingly important [90, 102]. In this context, *Process-Aware Information Systems* (PAIS) offer promising perspectives by enabling companies to model their business processes in terms of *process schemas* as well as to create, execute and monitor *process instances* based on such schemas in a controlled and efficient manner [123, 169]. Accordingly, a PAIS separates process logic from application code by means of explicit *process schemas*. Usually, the latter correspond to a graphical and abstracted representation of a real-world business process.

Business Process

Process-Aware Information System

Process Schema

Process Instance

Recently, *process modeling*, i. e., the practice of documenting business processes by process schemas, was ranked among the top usage scenarios for conceptual modeling in companies [36]. When modeling a business process it needs to be viewed from a number of different perspectives [61, 123, 157]. The *control-flow perspective* describes the ordering of the *activities* (i. e., process steps) of a business process as well as related execution constraints and decisions. In turn, the *operational perspective* relates activities to the application services (e. g., user forms or web services) to be invoked when executing the respective activity. Moreover, the *data perspective* connects activities and related application services with required data. The *resource perspective* provides a link between process elements (e. g., activities) and the given organizational structure, e. g., by associating process activities with user roles or other organizational entities. Finally, the *time perspective* characterizes business processes along their temporal properties (e. g., activity durations) as well as the temporal constraints to be obeyed during process instance execution (e. g., deadlines).

Control-Flow Perspective

Time Perspective

Temporal Constraints

Although time constitutes a crucial factor in modern business life and the proper handling of *temporal constraints* is vital in many application domains (e. g., healthcare, engineering) [33, 82], contemporary PAISs lack a comprehensive support of the time perspective of business processes. In a business context, where even small delays might cause significant problems, this constitutes a severe limitation for companies as it is crucial for them to know the temporal properties of their business processes and to monitor the adherence of the respective temporal constraints.

Recently, the proper support of the time perspective has been identified as a key challenge for future PAIS technologies [27, 33, 45, 80, 101]. In this context a wide variety of temporal concepts like deadlines, minimum or maximum durations, maximum time lags, and schedules need to be supported by the PAIS during process execution. Although there exists considerable work with respect to specific time-support features for PAISs, there is no comprehensive understanding of the time perspective of business processes as a whole [82]. In particular, it becomes necessary to understand what kind of information about the time perspective of a business process is actually required to provide proper support. Only then it becomes possible to comprehensively model business processes in terms of *time-aware process schemas*, which may then be executed and monitored by a *time-aware PAIS* in a robust way.

Time-Aware Process

As a prerequisite for a robust and error-free process execution in PAISs respective process schemas need to be sound [123, 171]. Informally, soundness can be described as the combination of three basic characteristics of a process schema: (i) the option to complete, (ii) proper completion, and (iii) absence of dead activities [123]. Regarding time-aware process schemas, the option to complete particularly requires that the *consistency of the temporal constraints* can be ensured [11, 29, 44]. The latter refers to the ability of executing a process schema without violating any of its temporal constraints. This is particularly challenging as temporal inconsistencies may be caused due to complex interactions among the temporal constraints. As a consequence, a precise understanding of temporal constraints is indispensable to be able to detect and analyze such interactions.

Temporal Consistency

As another challenge PAISs need to be flexible in order to cope with unforeseen events during run time [123, 144, 160]. To meet this demand, *adaptive PAISs* have been developed. In particular, they allow for dynamic adaptations of process instances during run time [5, 32, 123, 168]. Obviously, this run-time flexibility must be provided for time-aware processes as well [123, 143]. In particular, process flexibility even becomes more challenging if temporal constraints need to be obeyed by the process, as time can neither be slowed down nor stopped. Moreover, as process execution does not always stick to the plan, for example, it is common practice that deadlines have to be re-scheduled or processes have to be dynamically adapted in order to successfully and timely complete a process instance.

Adaptive PAIS

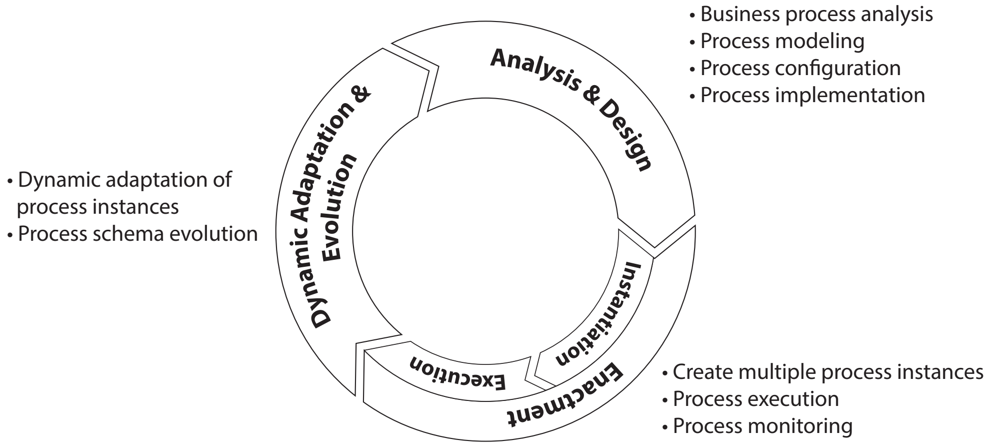

• Business process analysis
• Process modeling
• Process configuration
• Process implementation
Analysis & Design
Dynamic Adaptation & Evolution
• Dynamic adaptation of process instances
• Process schema evolution
Instantiation
Execution
Enactment
• Create multiple process instances
• Process execution
• Process monitoring

Figure 1.1: The process life cycle

1.1 Problem Statement

So far, time support has been rather limited in contemporary PAISs. As proper support of the time perspective is a fundamental prerequisite for the widespread use and further maturation of PAISs [27, 45, 101, 123], it needs to become an integral part of the entire *process life cycle* (cf. Figure 1.1). In the context of this thesis, the life cycle of a process is abstracted to three phases: *Analysis & Design*, *Enactment*, and *Dynamic Adaptation & Evolution* [123].

Process Life Cycle

During the *Analysis & Design* phase the real-world business process is analyzed and a process schema capturing its various perspectives is created [123]. To be able to automate the respective business process by a PAIS, at *design time* it needs to be modeled as completely and comprehensively as possible [14]. In turn, this requires high expressiveness of the used process modeling language [158]. As aforementioned, however, time support is limited in contemporary PAISs. In particular, current process modeling languages do not allow for the comprehensive modeling of the time perspective of business processes [77, 82]. Example 1.1 illustrates the diversity and complexity of the temporal aspects required for fully capturing business processes and their time perspective (adopted from [82]):

Analysis & Design Phase

Design Time

Example 1.1 (Patient Treatment Process)
Consider the simplified patient treatment process depicted in Fig. 1.2. First, a doctor orders a medical procedure for his patient. Then, the responsible nurse makes an appointment with the department (e.g., radiology) the procedure shall take place at. Before the actual treatment takes place, the patient needs to be informed about the procedure and be prepared for it. Moreover, shortly before starting the treatment, a specific preparation is required. After the treatment, the responsible doctor writes a short report if requested. Finally, aftercare is provided and the doctor responsible for the treatment creates a final medical report, which is then added to the patient record.

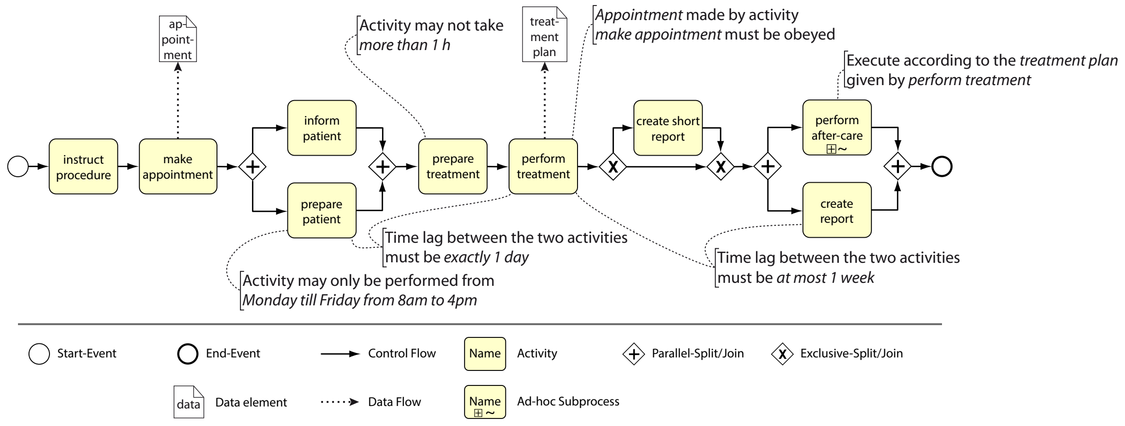

ap-point-ment
Activity may not take more than 1 h
treat-ment plan
Appointment made by activity make appointment must be obeyed
Execute according to the treatment plan given by perform treatment
instruct procedure
make appointment
inform patient
prepare patient
prepare treatment
perform treatment
create short report
perform after-care
create report
Time lag between the two activities must be exactly 1 day
Activity may only be performed from Monday till Friday from 8am to 4pm
Time lag between the two activities must be at most 1 week
Start-Event
End-Event
Control Flow
Name
Activity
Parallel-Split/Join
Exclusive-Split/Join
data
Data element
Data Flow
Name
Ad-hoc Subprocess

Figure 1.2: Simplified treatment process (adopted from [82])

When considering the time perspective of this rather simple process, a number of temporal constraints can be observed:

1. The *appointment* of the treatment, which is determined during the execution of activity make appointment, needs to be observed during process enactment.
2. The patient needs to be prepared *exactly 1 day before* the actual treatment takes place. Moreover, the preparation of the patient may only be accomplished during the opening hours of the anesthesia department, i. e., from *Monday till Friday between 8 am and 4 pm*.
3. Due to a tight schedule to be obeyed for the treatment room, preparation of the treatment must *not take more than 1 hour*; otherwise, subsequent treatments will be delayed.
4. Activity create report needs to be *completed no later than 1 week after* completing activity perform treatment.
5. During the execution of activity perform aftercare, different drugs are given to the patient according to the *treatment plan* defined by activity perform treatment. Such a treatment plan may state, for example, that drug A shall be administered every day at 8 am, 1 pm, and 6 pm, and drug B every two hours except if drug A is administered within the same hour.

To support the implementation of this process and its time perspective, a variety of temporal concepts (e. g., appointments, time lags between activities, durations) need to be covered by the PAIS and, hence, be expressible with the process modeling language used. Existing process modeling languages, however, only provide rudimentary support for the time perspective [82].

To properly address this gap, it needs to be determined which kinds of temporal concepts are required to fully cover the time perspective of business processes in a time-aware process schema. In this context, it is crucial to consider the relationship of the time perspective with other relevant process perspectives. For example, the *appointment* made by activity make appointment and later used for the treatment will generally be stored as process data, linking the time and data perspectives.

Run Time

After capturing a business process in a time-aware process schema, the correctness and robustness of the latter needs to be verified in order to ensure a robust and error-free execution of corresponding process instances at *run time*. Regarding time-aware processes this necessitates the temporal consistency of the process schema, i. e., the ability to successfully execute corresponding instances without violating any of their temporal constraints. Compared to other process perspectives, however, verifying the consistency of the time perspective is significantly more complicated as the interactions among the various temporal constraints of a time-aware process schema might result in complex interdependencies and hidden effects as shown by Example 1.2.

Example 1.2 (Interactions between Temporal Constraints)
Reconsider the process schema from Figure 1.2 and related temporal aspects as discussed in the context of Example 1.1. First, note that the preparation of the patient needs to be done exactly one day before the treatment. Moreover, the date of the treatment is fixed by the *appointment* made during activity make appointment. Hence, the preparation needs to be scheduled in accordance with the *appointment* for the treatment; i. e., the date when the preparation takes places is determined by this *appointment* as well. At the same time, the preparation of the patient may only be done from *Monday till Friday between 8 am and 4 pm*. Consequently, the treatment may only be scheduled from *Tuesday to Saturday* as otherwise one of the temporal constraints cannot be satisfied. In turn, the latter has to be taken into account when making the *appointment* for the treatment during activity make appointment.

Note that complex interactions among different temporal constraints might lead to scenarios in which it is impossible to satisfy all temporal constraints of a time-aware process schema at the same time. However, such scenarios need to be detected and prevented to enable a robust and error-free execution of corresponding process instances. To be able to analyze such interactions and to detect inconsistencies, a formal semantics of the temporal concepts used for specifying the time perspective of a process schema is required. Moreover, such a formal semantics contributes to avoid ambiguities regarding the use of these temporal concepts. So far, such semantics has been given only implicitly as part of formalisms used to detect possible inconsistencies [11, 27, 45, 175] (cf. [86] for a comparison), but no comprehensive and explicit description of a formal semantics exists in literature. Moreover, respective formalisms only support a limited subset of the temporal concepts required for modeling business processes [82].

Enactment Phase

Upon completion of the *Design & Analysis* phase, the resulting time-aware process schema can be deployed to the execution environment of the PAIS. During the *Enactment* phase multiple process instances corresponding to a particular process schema may then be created and executed. At this point it is important to note that verifying the temporal consistency of a process schema solely at design time is neither sufficient nor fully possible [82]. In particular, during run time the actual activity durations of executed activities become known and decisions regarding the execution of the process instance (i. e., the execution paths to be taken) are made. In general, not all temporal constraints are fully known at design time. For example, the value of the *appointment* made for the treatment in the context of Example 1.1 only becomes known during run time, as it is specific for each process instance. Hence, for time-aware process instances it is necessary to continuously monitor and update their temporal constraints and to regularly re-verify temporal consistency [82]. The latter is particularly important to be able to detect or predict critical situations during run time (e. g., excessive delays). Moreover, it should be possible to predict execution times of future activities in order to provide PAIS users with sufficient information to make well informed decisions regarding the process instance at hand as well as the other process instances concurrently executed by the PAIS.

Dynamic Adaptation & Evolution Phase

Despite all coordination efforts, process instances might run into difficulties as process execution does not always stick to the plan. Moreover, unforeseen events might occur during run time requiring a process instance to deviate from its predefined course of action. Consequently, process execution needs to be flexible to cope with such events [123]. In particular, it should be possible to dynamically adapt process instances during run time (e. g., to add, delete or move process activities) to appropriately react to such incidents. As business processes will evolve over time, additionally, it might become necessary to propagate respective changes to the corresponding process schema in order to ensure that the real-world process and the corresponding process instances remain aligned [123, 124]. Particularly in the context of long running processes, this necessitates the migration of already running process instances to the new process schema version. Generally, both *dynamic adaptations* of process instances [119] as well as *process schema evolution* [127] are considered to be part of the *Dynamic Adaptation & Evolution* phase. In this context, it is crucial to guarantee for a robust execution of the changed process instances in order to ensure their successful completion. Regarding time-aware processes this particularly includes the temporal consistency of the process instance. Thus, it must be possible to ensure that a dynamically modified process instance remains temporally consistent.

1.2 Research Contribution

Adaptive Time- and Process-Aware Information Systems

This thesis aims to provide fundamental concepts and techniques for the development of *Adaptive Time- and Process-Aware Information Systems* (ATAPIS). To foster time support in PAISs it contributes generic concepts, techniques and algorithms for modeling time-aware process schemas, for verifying their temporal consistency, and for executing as well as dynamically modifying corresponding process instances. The main research contributions of the thesis can be summarized as follows:

1. A well-founded set of *time patterns* representing temporal concepts commonly found in business processes is presented. Respective time patterns are based on empirical evidence gained from a comprehensive analysis of business processes and related work on time support in PAISs. The time patterns provide a universal and comprehensive set of notions for describing temporal aspects of business processes and for eliciting fundamental requirements. Moreover, they provide a profound foundation for analyzing and comparing PAISs with respect to their ability to deal with temporal aspects of business processes.

2. A *precise formal semantics* for each of the time patterns is provided. Respective semantics are based on an in-depth analysis of a large set of industrial and scientific use cases. To foster the use of the time patterns in a wide range of application scenarios, their semantics are defined independent of a specific process modeling language (e. g., BPMN) or process modeling paradigm (e. g., declarative vs. imperative). At the same time, the formal semantics of the time patterns closely consider the relationship of the time perspective with other process perspectives, including the control flow and data perspectives. The definition of their formal semantics will foster the integration of the time patterns into PAISs, significantly broadening the application scope of the latter.

3. A generic *framework* for implementing the time patterns is proposed. This framework allows verifying the temporal consistency of time-aware process schemas based on the defined pattern semantics. Subsequently, the framework is extended to consider advanced aspects, including the contingent nature of activity durations as well as the challenges faced when considering alternative execution paths.

4. A theoretical framework as well as an algorithm for ensuring the temporal consistency of time-aware process instances during run time are presented. The corresponding concepts specifically consider the facts that (a) activity durations are contingent and the actual duration of an activity instance only becomes known after its completion; (b) certain temporal constraints (e. g., appointments) might not be known at design time, but can solely be determined at run time; and (c) execution decisions made during run time may have significant impact on the temporal properties of the remainder of the process instance.

5. A set of change operations are presented, which are able to dynamically modify a time-aware process instance while preserving its temporal consistency. Moreover, we provide a detailed analysis of the effects respective change operations may have on the temporal constraints of the process schema. Based on this, a technique for approximating the resulting temporal properties of a modified process instance is proposed. In particular, this technique can be used to significantly reduce the complexity of the calculations required when applying multiple change operations at the same time. The latter is particularly advantageous in the context of process

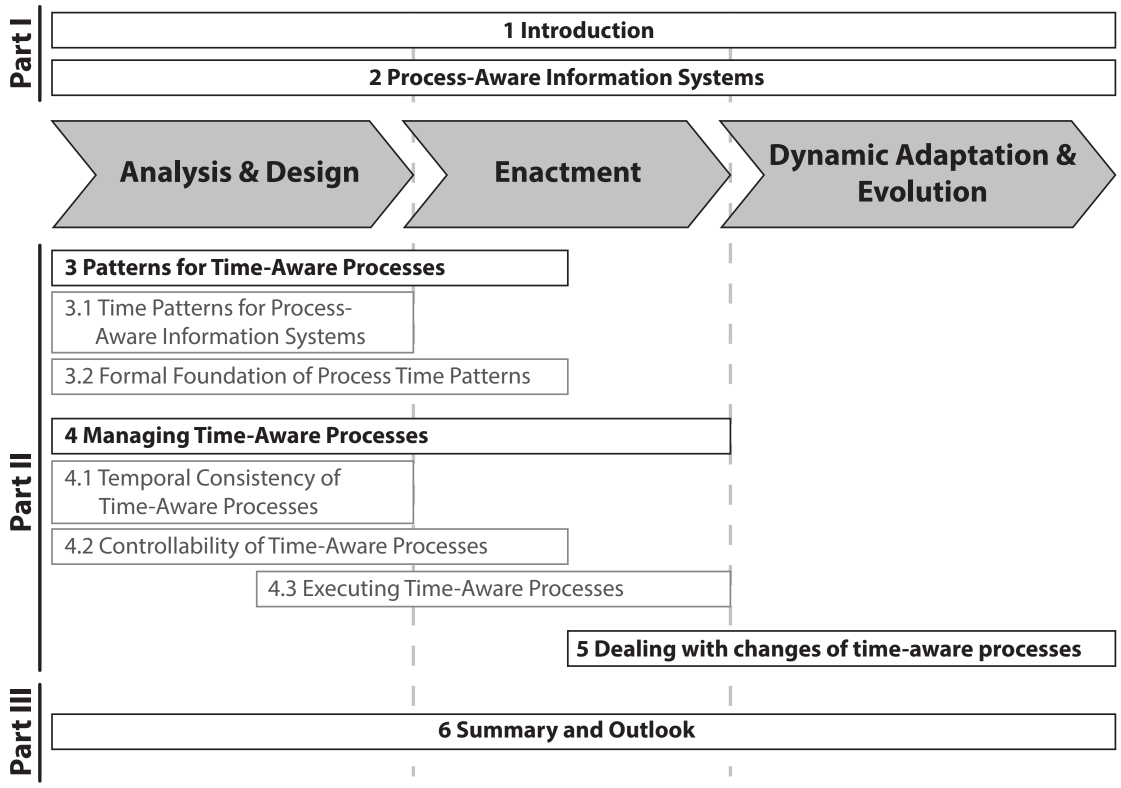

Part I
1 Introduction
2 Process-Aware Information Systems
Analysis & Design
Enactment
Dynamic Adaptation & Evolution
Part II
3 Patterns for Time-Aware Processes
3.1 Time Patterns for Process-Aware Information Systems
3.2 Formal Foundation of Process Time Patterns
4 Managing Time-Aware Processes
4.1 Temporal Consistency of Time-Aware Processes
4.2 Controllability of Time-Aware Processes
4.3 Executing Time-Aware Processes
5 Dealing with changes of time-aware processes
Part III
6 Summary and Outlook

Figure 1.3: Outline

schema evolution, where possibly hundreds or thousands of process instances may have to be migrated to a new process schema version.

Altogether, the results presented in this thesis aim at the profound integration of the time perspective into contemporary PAISs. In the context of the ATAPIS[1] project, moreover, a comprehensive prototype has been developed demonstrating the practical feasibility as well as applicability of the main concepts of this thesis.

1.3 Outline

Figure 1.3 summarizes the outline of this cumulative thesis and classifies each part along the process life cycle. The thesis consist of three main parts.

Part I motivates the need for properly supporting the time perspective of business processes in PAISs (Chapter 1) and provides relevant background notions in Chapter 2.

[1]**A**daptive **T**ime- and **P**rocess-Aware **I**nformation **S**ystem; visit: http://dbis.info/atapis

Part II summarizes the scientific contribution achieved by this thesis. In particular, this cumulative dissertation consists of five main publications denoted as (LWR10) [77], (LWR14) [82], (LR14) [71], (LPCR13) [79], and (LRW16) [86] complemented by several technical reports [70, 72, 73, 80] providing additional details. Each publication has had significant impact on the state-of-the-art in PAISs and has been published in renowned journals or peer-reviewed conferences.

Chapter 3, and in particular Section 3.1, introduce the time patterns. The latter where originally introduced in (LWR10) and later extended and described more in-depth in (LWR14). They form the foundation for comprehensively representing and modeling temporal aspects of business processes. Section 3.2 discusses the formal semantics of the time patterns originally elaborated and presented in (LRW16).

Chapter 4 deals with the management of time-aware processes in PAISs. First, Section 4.1 presents the ATAPIS framework for verifying the temporal consistency of time-aware processes as discussed in (LRW16). In turn, Section 4.2 extends the ATAPIS framework to properly deal with the contingent nature of activity durations. Furthermore, it considers alternative execution paths more closely (cf. LPCR13). Finally, Section 4.3 considers the execution of time-aware process instances in PAISs as discussed in (LPCR13) as well as (LRW16).

Chapter 5 deals with the dynamic modification of time-aware process instances. Particularly, the time-aware change operations, which were first discussed in (LR14), are presented and an approach for improving the support of dynamic changes of time-aware processes is considered.

Part III and Chapter 6 conclude the thesis. Section 6.1 summarizes its contribution. Moreover, Section 6.2 discusses other publications the author of the thesis co-authored and which are related to time and processes. Finally, Section 6.3 closes with an outlook on various aspects related to time support in PAISs that might be addressed by future research.

2 Process-Aware Information Systems

This chapter summarizes basic concepts and notions needed for understanding this thesis.

The thesis deals with business processes and Process-Aware Information Systems (PAISs). A *business process* can be defined as a set of connected business activities which collectively realize a particular business goal [169] (e. g., patient treatment, car repair, claim handling). *Process-Aware Information Systems*, in turn, constitute a special kind of *Enterprise Information Systems* (EIS) enabling the computer-aided support of business processes [42, 123]. Compared to other kinds of EISs, a PAIS is characterized by the separation of process logic from application code—providing an additional architectural layer to the EIS. To this end, at design time the process logic is explicitly specified in terms of a process schema. Respective process schemas may then be deployed to the execution environment of the PAIS. At run time, a PAIS may create multiple process instances based on a process schema and execute them according to the defined logic. Moreover, PAISs enable the integration of application services, data, users and other resources. Usually, the core of a PAIS is build by a process management system, which provides required services for process modeling, execution, monitoring, and user interactions.

Business Process

Process-Aware Information System

2.1 Process Schema

For each business process to be supported by a PAIS, a *process schema* has to be specified based on the constructs provided by a process modeling language. In particular, a process schema describes the flow of work of a particular business process. This is achieved by

Process Schema

specifying the flow of business artifacts (e. g., activities, events) within the process as well as the data flow between them.

In general, process schemas may be specified using different formalisms (e. g., graph-based, constraints-based, or based on logic formulas). So far, graph-based process modeling languages have gained a broad acceptance in literature and commercial tools. Following this, the thesis presumes that a process schema corresponds to a directed graph, which comprises a set of nodes and a set of control edges between them (cf. Figure 2.1). Nodes may represent *activities*, *process events* and *control connectors*. In turn, *control edges* specify precedence relations between nodes as well as loop-backward relations.

Control Edge

This thesis uses BPMN (Business Process Modeling and Notation) [110] for visualizing and describing processes (cf. Figure 2.1). Due to its standardization [110], BPMN has gained wide acceptance in science and industry. Note that, even though we use BPMN for illustration purpose, the described concepts are not language-specific, i. e., they can be integrated in any process modeling language supporting the basic concepts discussed in the following. Moreover, we use additional symbols not being part of BPMN to illustrate temporal constraints.

Activity

Activities may either be atomic or complex. An atomic activity is usually associated with an application service and corresponds to a human task—requiring user interaction—or an automated task. In turn, a complex activity refers to a subprocess. The latter allows for the hierarchical decomposition of processes. In the context of this thesis, we consider complex activities as self-contained (i. e., there is no direct relationship between a subprocess and the respective parent process), and, therefore, in most cases do not explicitly differentiate between atomic and complex activities.

Process Event

A *process event* represents something happening during process execution, which is not bound to an activity (e. g., receipt of a message). Note that, as the term event is very general and may describe many things in a process, we differentiate two types of events. While a *process event* is explicitly specified in the process schema (cf. Figure 2.1), other events are implicitly triggered during process execution (e. g., start/end event of an activity instance; cf. Section 2.2)

Control Connector

Control connectors are used to express splits and joins in the control flow of a process. In the context of this thesis, we specifically consider the following control flow patterns [158]: sequence, parallel split (*AND-split*), synchronization (*AND-join*), exclusive choice (*XOR-split*), simple merge (*XOR-join*), and *structured loops*. Note that these patterns constitute the core of any process modeling language and cover most processes found in practice [109]. Moreover, note that the presented concepts are not limited to these patterns. However, we do not explicitly consider the other patterns to keep respective discussions compact.

To simplify the analysis of process schemas we further assume that the start and end nodes of a structured loop are distinct from normal XOR-split and XOR-join nodes; i. e., there is an explicit loop construct in the process modeling language.[1] To graphically

[1]Note that this does not apply to BPMN causing additional complexity when analyzing processes.

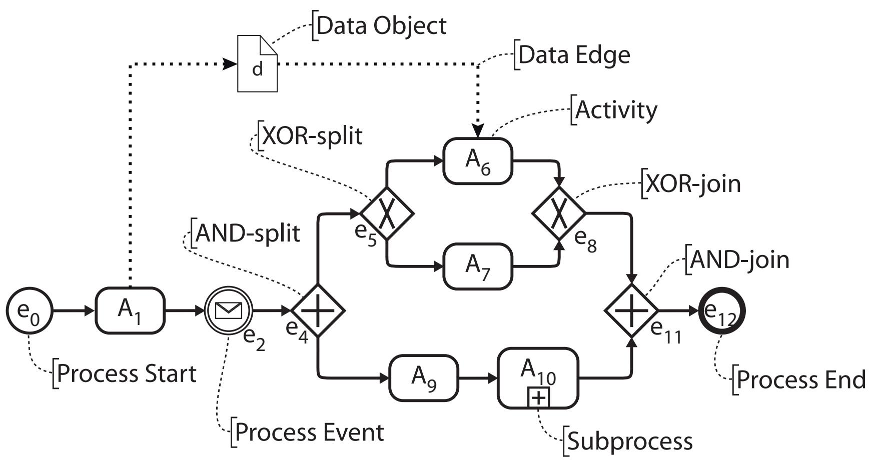


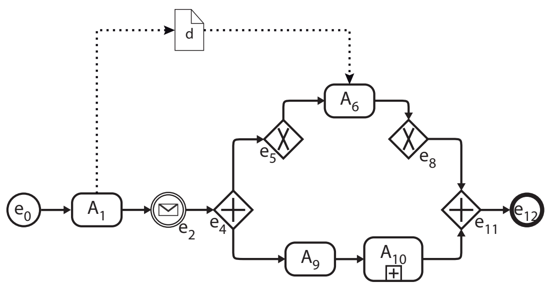
Figure 2.1: Core concepts of a process modeling language

distinguish between loop-blocks and XOR we use the exclusive gateway symbol with an "X" to represent an XOR-split/-join and the symbol without an "X" to represent loop-start and loop-end nodes. Finally, we assume that loops are well-nested, i. e., they may be nested, but must not overlap.

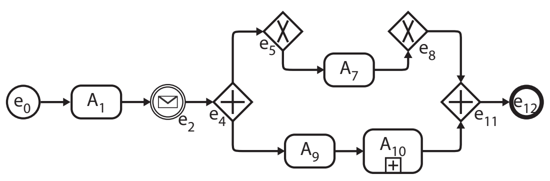
The thesis uses the notion of *activity set* to refer to a subset of the activities of a process schema. The elements of an activity set do not have to comply with any structural requirement. When referring to specific regions of a process schema, in turn, we use the notion of *process fragment*. To be more precise, a *process fragment* refers to a sub-graph of a process schema with single entry and single exit node (also denoted as single-entry, single-exit region; i. e., *SESE-region*).

In addition to the described control flow elements, a process schema may contain process-relevant *data objects* as well as *data edges* linking activities with data objects (cf. Figure 2.1). More precisely, a data edge either represents a read or write access of the referenced activity to the referred data object.

Example 2.1 (Process Schema)
Fig. 2.1 shows a process schema consisting of five activities, one process event and four control connectors: Activity A_1 is succeeded by process event e_2, which, in turn, is succeeded by the parallel execution of either activity A_6 or A_7, and activities A_9 and A_{10}. Activities A_1 to A_9 are atomic, whereas A_{10} constitutes a complex activity; i. e., a sub-process with its own process schema. The region of the process schema containing activities A_6 and A_7 together with the depicted control connectors (i. e., the XOR-split e_5 and XOR-join e_8) constitutes an example of a process fragment (i. e., a SESE-region). Finally, any non-empty subset of activities A_1, ..., A_{10} constitutes an activity set.



(a) Execution path (1)



(b) Execution path (2)

Figure 2.2: Execution paths of Figure 2.1

Execution Path

If a process schema contains one or more XOR-splits or loops, not all process instances perform exactly the same set of activities. The concept of *execution path* allows identifying which activities and control connectors are performed during the execution of a process instance. In particular, an execution path of a process schema denotes a connected maximal subgraph of the process schema—containing its Start- and End-nodes—, in which all XOR-split connectors have exactly one branch and each loop block has a fixed number of repetitions. Each execution path represents one possible execution of the respective process schema.

Example 2.2 (Execution Path)
Consider the process schema from Figure 2.1. The schema comprises two execution paths as depicted in Figure 2.2, i. e., one execution path where e_2 is followed by the execution of A_6 in parallel with A_9 and A_{10} (i. e., for XOR-split e_5 the upper path is selected), and another execution path where e_2 is followed by the execution of A_7 in parallel with A_9 and A_{10} (i. e., the lower path is chosen).

Syntactical Correctness

Soundness

A fundamental prerequisite for the successful execution of a process schema is its correctness. First of all, a process schema must be *syntactically correct*, i. e., it must match the grammar rules defined by the respective process modeling language. However, syntactical correctness is not sufficient to guarantee correct executability of process instances. Another fundamental property to ensure correct executability is the *soundness* of the process schema. Informally, soundness can be described as the combination of three basic characteristics of a process schema [123, 171]:

Option to Complete

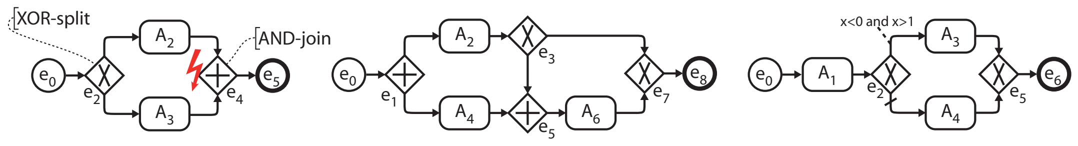
(i) *Option to complete*: Once started, a corresponding process instance must always be able to complete.

Proper Completion

(ii) *Proper completion*: When a process instance completes, there must not be any activated or still running activity.

Absence of Dead Activities

(iii) *Absence of dead activities*: There must be no dead activities that may never become enabled.



(a) Deadlock (b) Proper completion (c) Dead activity

Figure 2.3: Process schema soundness – Anti-Patterns

The *option to complete* guarantees that every process instance is able to successfully complete. Moreover, *proper completion* ensures that once a process instance signals its termination, there are no activities which are left still running. Note that the latter might result in an undefined behavior. Finally, *absence of dead activities* ensures that there are no unreachable parts in the process schema, which, in turn, generally indicates an error in the process schema. Example 2.3 illustrates these three properties along different anti-patterns [123].

Example 2.3 (Process schema soundness – Anti-Patterns)
Figure 2.3 (a) shows a process schema exhibiting a deadlock. In particular, for the XOR-split e_2 only one of the outgoing branches is selected. However, AND-split e_4 will wait for the completion of both incoming branches. This, in turn, means that an instance of the process schema will never be able to complete (i. e., it violates the *option to complete* property).

Figure 2.3 (b) shows a process schema violating the *proper completion* property. In particular, if after the completion of activity A_2, at the following XOR-split e_3, the upper path is chosen, the process instance might reach the Process-End event (and thus complete), while activity A_4 is still active (i. e., running).

Finally, the process schema depicted in Figure 2.3 (c) contains a dead activity as the branch condition "$x < 0$ and $x > 1$" of the upper branch always evaluates to false. Generally, this indicates an error in the process model as such behavior is undesirable.

Different frameworks and tools have been developed for checking soundness of a process schema [116, 150, 155], for example, based on techniques like reachability analysis and model checking. Finally, different correctness criteria for the data perspective have been defined as well (e. g., no missing data, no lost updates) [123, 150, 153].

2.2 Process Execution

A process schema describes the workflow of a business process. To enable process execution by a PAIS, most process modeling languages provide an operational semantics

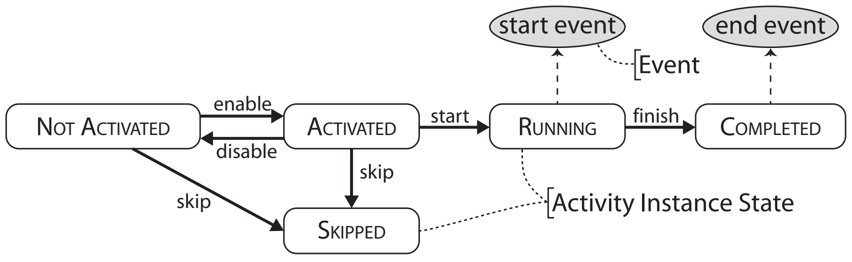


Figure 2.4: Activity life cycle and relevant events

that defines how a process schema shall be processed during execution (for BPMN see [110]).

Process Instance

For each business process to be executed a *process instances* is created by the PAIS and executed according to its predefined process schema [169]. At run time, the nodes of a process instance are processed and executed according to the order defined by the control flow of the corresponding process schema and its underlying operational semantics. In this context, *activity instances* represent an execution of single process step (i. e., an activity) of a particular process instance.

Activity Instance

We assume that activity instances are executed according to the *activity life cycle* depicted in Fig. 2.4 [123]. When a process instance is started, all its activities are in state NOT ACTIVATED. As soon as an activity may be executed, its state switches to ACTIVATED. When a user starts an activated activity instance, its state switches to RUNNING. As soon as the user finishes work, the state of the activity instance switches to COMPLETED. Finally, state SKIPPED is assigned to non-executed activities (e. g., activities of an outgoing path of an XOR-split not selected during run-time).

When an activity instance is started, the data values provided by associated data objects through a read access are consumed. In turn, when completing an activity instance, data objects linked to the activity through a write access are provided.

Event

During the execution of a process instance, different *events* are triggered. For example, when creating a new process instance, a corresponding start event is generated. Upon its completion, in turn, an end event is generated. Moreover, each node belonging to the process generates a start/end event when starting/completing it (cf. Figure 2.4). Finally, events may be triggered by external sources as well (e. g., receipt of a message). We use the notion of event as general term for something happening during process execution.

Exection History

Execution Trace

Generally, a process instance is associated with an *execution history* capturing all events that occurred during its execution so far (e. g., all activities and gateways executed). Such an execution history is also referred to as *execution trace*. Note that by “replaying” the execution trace on the respective process schema the current execution state of the process instance can be reconstructed.

Example 2.4 illustrates these concepts.

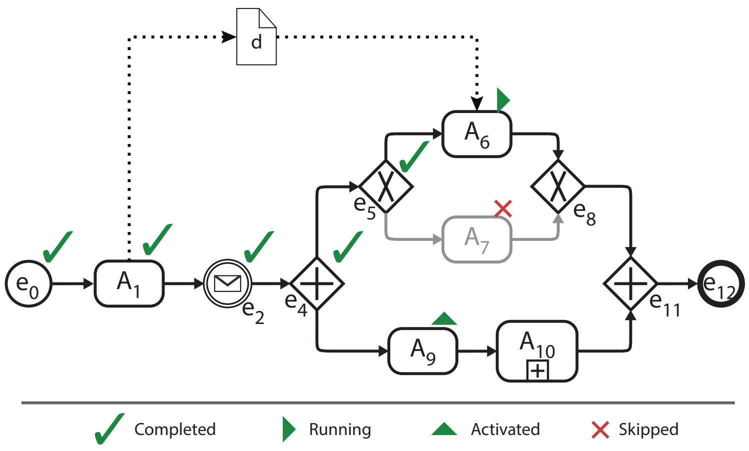


Figure 2.5: Process instance

Example 2.4 (Process Instance)
Figure 2.5 depicts a running process instance of the process schema from Figure 2.1. One activity (i. e., A_1) was executed previously. Moreover, event e_2 occurred and XOR-split e_5 was evaluated. Activity A_6 is currently RUNNING. This implies that the lower branch of XOR-split e_5 was deselected and activity A_7 marked as "SKIPPED". Finally, activity A_9 is in state ACTIVATED.

Informally, the execution trace of this process instance comprises the following events:

- Start event e_0 of the process instance.
- Start and completion events $e_{\mathsf{A}_1 S}/e_{\mathsf{A}_1 E}$ of activity A_1.
- Event e_2.
- Execution of AND-split gateway e_4 and XOR-split gateway e_5.
- Start event $e_{\mathsf{A}_6 S}$ of activity A_6.

Finally, the thesis uses the notion of *process instance set* to refer to a set of process instances executed by a PAIS. These process instances, in turn, may either be enacted based on the same process schema, but may also run on different process schemas.

2.3 Process Flexibility

PAISs need to be flexible to properly deal with exceptions, changes in the environment, and evolving business processes. Generally, process flexibility may be achieved in three different ways [123]: (a) through pre-specified flexibility (i. e., flexibility-by-design), (b) by allowing for loosely specified process schemas, which may be refined during run time, and (c) through structural process changes. Whilst flexibility-by-design is usually easy to accomplish (e. g., by using an exclusive choice), loosely specified process schemas and flexibility-through-changes require special support by the PAIS. Note that this thesis focuses on flexibility-by-design and flexibility-through-change, but most of the presented concepts may be applied in the context of loosely specified processes as well (e. g., consider [6]).

Generally, structural process changes may be applied on both the process schema and process instance level [123]. In particular, as business processes evolve over time, it becomes necessary to also evolve respective process schemas, in order to keep the process schema aligned with the real world. This is denoted as *process schema evolution*. Such schema evolution additionally necessitates the propagation of respective changes to already running process instances (e. g., in the context of long running processes).

Process Evolution

In turn, structural process changes of single process instances are usually performed to deal with exceptions or unanticipated changes in the environment [167]. The effects of such *dynamic process changes* are usually instance-specific and do not affect other process instances concurrently being executed [167]. Hence, they result in an instance-specific process schema.

Dynamic Process Change

Regarding dynamic changes and process evolution an important aspect concerns the behavioral and structural soundness of the modified process instance and the corresponding (instance-specific) process schema. Structural soundness of a modified process schema may be verified based on general correctness notions of process schemas (cf. Section 2.1) [122]. In turn, a widespread correctness notion for behavioral soundness is *state compliance* [18, 130]. Informally, a process instance I based on process schema S is *state compliant* with modified process schema S' derived from S, if the current execution trace σ of I (cf. Definition 3.1) is reproducible on S' [123]. In, [133] the notion of state compliance is extended to be applicable in the context of loops as well. Moreover, state compliance can be used to decide whether a particular change may be correctly applied to a process instance in its current state as illustrated by Example 2.5.

State Compliance

Example 2.5 (Process Flexibility)
Figure 2.6 depicts three process instances I_1, I_2 and I_3 to which different changes shall be applied.

Instance I_1 shall be dynamically modified by deleting activity A_7 and inserting activity X after XOR-join e_8. This is possible as activity A_7 has been skipped and the part of the process where X shall be inserted has not been executed yet.

In turn, I_2 shall be dynamically changed by inserting activity Y between activities A_9 and A_{10}. Again this is possible, although activity A_{10} is already in state Activated. However, in this case the state of the instance has to be adapted after the change. In particular, activity A_{10} switches to state Not Activated and, in return, Y immediately becomes Activated. This can be also seen when replaying the respective execution trace from original instance I_2 on the instance-specific process schema of modified instance I_2'.

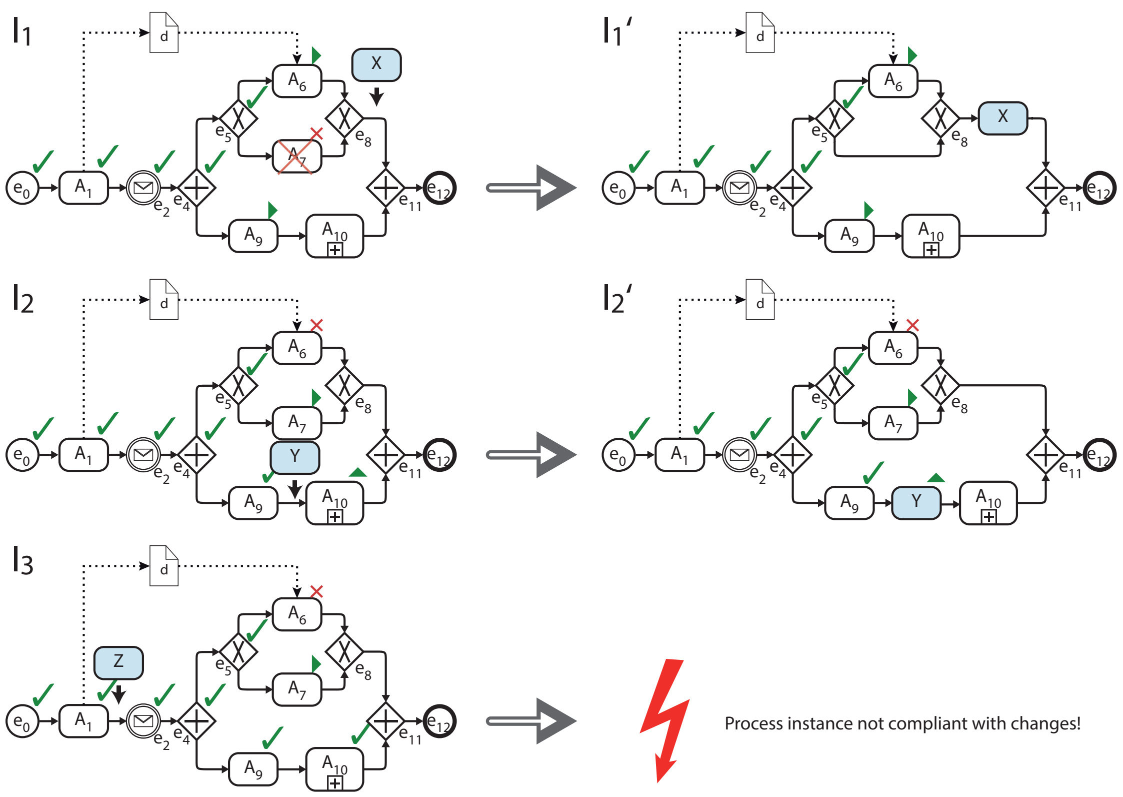


Figure 2.6: Dynamic process instance changes

Finally, for instance I_3, Z shall be inserted between activity A_1 and process event e_2. However, this is not possible as the process instance has already progressed too far. This can be seen when trying to replay the corresponding execution trace on the modified process schema. In particular, the trace cannot be replayed as the newly inserted activity can not be marked as RUNNING or COMPLETED.

Part II

Time-Aware Process Support

3 Patterns for Time-Aware Processes

Time is what we want most, but what we use worst.

(William Penn, 1644–1718)

3.1 Time Patterns for Process-Aware Information Systems

The content of this section was first published as follows:

(LWR10) A. Lanz, B. Weber, and M. Reichert. Workflow time patterns for process-aware information systems. In *Enterprise, Business-Process and Information Systems Modeling — Proceedings of the 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010, held at CAiSE 2010*, volume 50 of *Lecture Notes in Business Information Processing*, pages 94–107. Springer, June 2010. DOI: 10.1007/978-3-642-13051-9_9

A significantly extended version of this work was published as follows:

(LWR14) A. Lanz, B. Weber, and M. Reichert. Time patterns for process-aware information systems. *Requirements Engineering*, 19(2):113–141, 2014. DOI: 10.1007/s00766-012-0162-3

The original articles are added to Appendices A.1 and A.5, respectively.

Preliminary results as well as additional details on selected aspects have been published in a technical report as well [73].

3.1.1 Problem Description

As motivated in Chapter 1, both the specification and operational support of temporal constraints constitute fundamental challenges for any PAIS. To support the design and implementation of time-aware processes, a variety of temporal concepts (e. g., deadlines, minimum and maximum time lags, durations, and schedules) need to be supported by the PAIS. Therefore, respective concepts need to be supported by the used process modeling language and execution environment.

To discuss some of the challenges emerging in this context we consider Example 3.1.

Example 3.1 (Patient Treatment Process - Revisited)
Let us revisit the simplified treatment process as described in Example 1.1. The process is shown in Figure 3.1, which also depicts the temporal constraints introduced by Example 1.1.

First, consider the *appointment* for the treatment. For its proper support, it must be possible to attach a respective temporal constraint, which refers to the `date` provided by activity make appointment, to activity perform treatment. Usually, an appointment is considered to be the earliest start date of an activity. In turn, deadlines are used to refer to the latest end date of an activity. Note that both concepts (i. e., appointment and deadline) are similar and may thus be considered as different variants of the same temporal constraint. In general, a temporal constraint correlating with a fixed date may refer to the start or end of an activity, and may restrict the earliest or latest start/end of the activity. Finally, note that an appointment or deadline may change during the execution of a process instance. For example, in certain treatment scenarios it might become necessary to enable the performer of activity inform patient to postpone the treatment (i. e., change its appointment); e. g., due to a bad physical shape of the patient.

Second, consider the temporal constraint between activities prepare patient and perform treatment. It states that the time lag between these two activities shall be *exactly 1 day*. Again, one must consider whether such a time lag refers to the start or end of the respective activities. Moreover, such a constraint may restrict the minimum time lag (e. g., at least 1 day), the maximum time lag (e. g., at most 1 day), or both (e. g., between 1 and 2 days; exactly 1 day).

Third, the constraint that the preparation of the patient may only be accomplished during the opening hours of the anesthesia department corresponds to some kind of timetable or schedule, restricting the execution of this activity. In general, such schedules may be described by simple expressions based on a calendar (e. g., Mo–Fr, 8 am to 4 pm). Again they may either refer to the start or end of the activity (or both).

Finally, it is common practice that activities have assigned maximum duration limits (cf. activity prepare treatment), e. g., to avoid delays for subsequent activities or comply with business regulations. Moreover, minimum durations are frequently required for planning

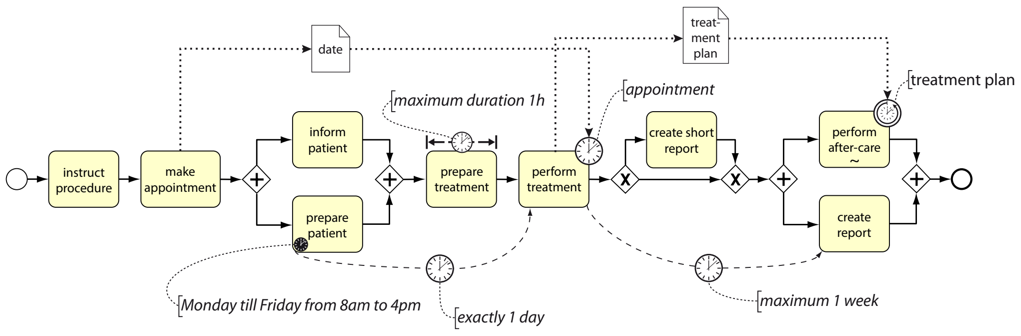


Figure 3.1: Simplified treatment process with temporal constraints[1] (LWR14)

purposes. Note that durations are not only relevant for activities, but for sets of activities and entire processes as well.

As can be seen from Example 3.1, a variety of temporal concepts need to be supported by a PAIS to enable the implementation and operational support of time-aware processes. Although there exists considerable work regarding the support of *time-aware processes* in PAIS, there is no comprehensive framework considering the specification of the *time perspective* in its entirety. To tackle similar challenges with respect to other process perspectives, a number of workflow patterns [134, 137–139, 141, 152, 158, 167] were introduced. Workflow patterns provide a means for analyzing the expressiveness of process modeling languages and tools regarding their support of these perspectives. Additionally, workflow patterns can be used for eliciting and analyzing requirements described by formal process specifications.

3.1.2 Scientific Contribution

To tackle the discussed challenges, this part of the thesis presents 10 *time patterns* representing temporal constraints commonly occurring in the context of time-aware processes. The time patterns where first introduced in (LWR10). In turn, (LWR14) extends this work providing an in-depth description of all time patterns. Moreover, a systematic literature review as well as an evaluation are conducted to assess the completeness and usefulness of the proposed time patterns.

To ground the time patterns on a solid basis, a *design science* approach [53, 54] is adopted for identifying and evaluating related patterns (cf. Figure 3.2). Selection criteria for the time patterns are “patterns covering temporal aspects relevant for the modeling and control of business processes and activities respectively” (LWR14), i. e., the identified time patterns focus on the relationship between the *time* and *control-flow perspective* of

[1]Note that we use an extension of BPMN to visualize time constraints in process schemas.

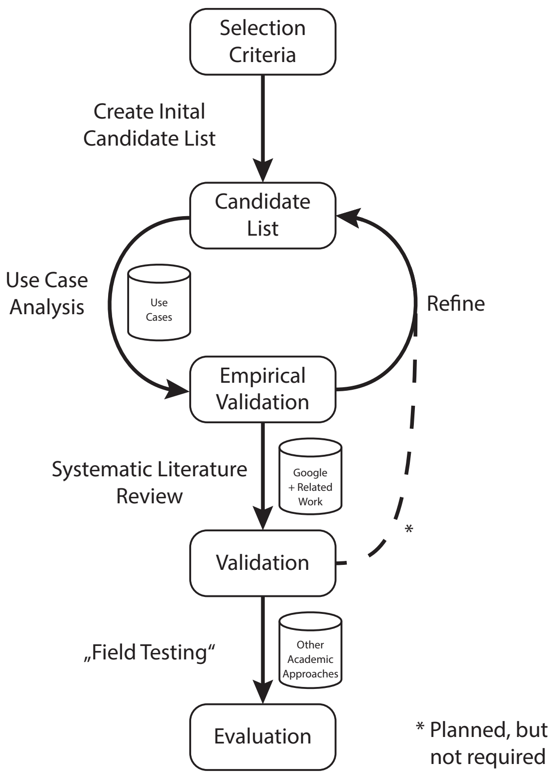


Figure 3.2: Research methodology applied in (LWR14)

processes, but do not specifically consider other perspectives like the resource and data perspectives.

To identify the time patterns, first of all, a list of candidate patterns is created based on the knowledge and experience of the (LWR14) authors in the field. Then, a large set of industrial cases, process schemas and process documentation consisting of more than 270 individual processes is analyzed to refine and extend the pattern candidate list as well as to provide empirical evidence for each of the time patterns (cf. (LWR14) for details). Finally, to keep the number of different patterns manageable, *design choices* are introduced to characterize *pattern variants*; i. e., design choices enable the parametrization of the time patterns and allow consolidating similar concepts into one pattern (e. g., appointment and deadline).

Design Choices

Pattern Variant

Time Patterns

Time Pattern

This research resulted in the 10 *Time Patterns* depicted in Table 3.1, which can be divided into categories based on their semantics. Each of the time patterns constitutes a solution for representing and realizing commonly occurring temporal aspects in PAISs. In (LWR14), these patterns are systematically described in detail, including synonyms,

Category I: Durations and Time Lags	
TP1	Time Lags between Activities
TP2	Durations
TP3	Time Lags between Events
Category II: Restricting Execution Times	
TP4	Fixed Date Elements
TP5	Schedule Restricted Elements
TP6	Time-based Restrictions
TP7	Validity Period
Category III: Variability	
TP8	Time-dependent Variability
Category II: Recurrent Process Elements	
TP9	Cyclic Elements
TP10	Periodicity

Table 3.1: Process time pattern catalogue (LWR14)

description, design choices, solution, context, examples from the data sources, and related patterns.

The three time patterns grouped in *Category I: Durations and Time Lags* provide support for expressing durations of and time lags between process elements. In detail, time pattern *TP1: Time Lags between Activities* allows defining time lags between two activities. Such a time lag may express a minimum or maximum temporal distance or a temporal interval between the two activities. Moreover, it may represent a start-to-start, start-to-end, end-to-start, or end-to-end relationship between respective activities. Note that time lags may not only be required between directly succeeding activities, but also between arbitrary activities presuming that these may be executed conjointly the the context of a particular process instance.

Time Lags between Activities

Using time pattern *TP2: Durations* the duration of process elements of arbitrary granularity can be specified as well as restricted, i. e., *TP2* may be used to specify the duration of a single activity or process as well as the duration of a set of activities or set of process instances. Moreover, a duration may correspond to a minimum duration, a maximum duration, or both (i. e., an interval).

Durations

Finally, time pattern *TP3: Time Lags between Arbitrary Events* enables the specification of a time lag between two arbitrary discrete events. The latter may be related to the execution of activities, but may also be triggered by an external source not controllable by the PAIS (e. g., receiving a message or events occurring in the physical world). This pays special attention to the fact that not everything that may happen during the execution of a process instance can be attributed to a specific activity. Accordingly, *TP3* constitutes a generalization of *TP1*. However, its use cases and semantics are quite different, motivating the introduction of a discrete pattern.

Time Lags between Arbitrary Events

Category II: Restricting Execution Times comprises four time patterns, which allow specifying constraints regarding possible execution times of single activities or processes. In particular, *TP4: Fixed Date Elements* allows specifying deadlines and fixed execution dates, i. e., *TP4* allows specifying that a particular activity or process instance must be started or completed before or after a particular date. In this context it is important to note that the corresponding date is specific to a process instance, i. e., it only becomes known during run time or when creating the respective process instance.

Fixed Date Elements

Schedule Restricted Elements

Time pattern *TP5: Schedule Restricted Elements* is required to express temporal aspects like opening hours or working times. In particular, it allows restricting the start and completion of an activity or process by a schedule. Such a schedule consists of a (possibly infinite) set of time slots during which the activity/process may be started or completed.

Time-based Restrictions

Pattern *TP6: Time-based Restrictions* allows restricting the number of times an activity, set of activities, process, or set of process instances may be executed within a given time frame. A particular variant of this pattern is a time-based mutual exclusion, i. e., two activities may be executed in arbitrary order, but cannot be executed at the same time or within a particular period of time of each other (e. g., because they require the same non-shareable resource). Regarding *TP6*, in a later work it was discovered that the initially specified *design choices* had to be extended by a new design choice indicating how the execution time of an activity and a given time frame (e. g., the execution time of another activity) shall be compared in order to fully capture the semantics of this pattern (cf. LRW16).

Validity Period

Time pattern *TP7: Validity Period* is similar to *TP4*, i. e., it allows expressing that an activity or process must not be started or completed before or after a particular date. As opposed to *TP4*, he particular date is not specific to a process instance, i. e., it is the same for all instances of the considered process schema. For example, this might be required to restrict the remaining life time of an obsolete process implementation and to schedule the rollout of the new schema version.

Time-dependent Variability

Pattern *Category III: Variability* consists of a single pattern, i. e., *TP8: Time-dependent Variability*. *TP8* allows specifying varying control flow depending on temporal aspects. That is, depending on temporal aspects, like the execution time of an activity or the time lag between two activities, different control flow paths may be chosen. Amongst others, this allows choosing an alternative path if no response to a message is received within a certain time frame.

Cyclic Elements

Category IV: Recurrent Process Elements comprises two patterns required for expressing temporal constraints in connection with recurrent activities or process fragments (i. e., loops). Particularly, pattern *TP9: Cyclic Elements* enables us to define time lags between activities contained within a loop structure. As opposed to *TP1*, *TP9* defines a time lag across different loop iterations, i. e., the source and target activity of the time lag belong to different iterations of the same loop. Note that *TP9* may not only restrict the time lag between activities of two directly succeeding loop iterations, but also between activities belonging to two arbitrary iterations. Similar to *TP1*, pattern *TP9* may represent a minimum time lag, maximum time lag, or interval. It further may describe a start-to-start, start-to-end, end-to-start, or end-to-end relationship between the two activities. Finally, it is noteworthy that the source and target of a cyclic element may actually refer to the same activity, creating a time lag between two subsequent instances of this activity.

Periodicity

Pattern *TP10: Periodicity* allows specifying periodically recurring sets of activities according to an explicitly defined periodicity rule. The latter describes the recurrence schema of the respective activity (e. g., every Monday and Wednesday at 11:30) as well as a particular exit condition. For example, this pattern is common in the healthcare domain. As opposed to *TP9*, pattern *TP10* emphasizes possible execution dates of recurrent activities, but not the time lag between them. Moreover, periodicity rules may involve more than two activities. Note that during run time, a *Periodicity* can be realized by the combined use of patterns *TP1-6*, *TP8* and *TP9*. Since respective periodicity rules generally become known only during run time, however, no pre-specification of a corresponding process fragment is possible. Therefore, *Periodicity* as an additional layer of abstraction is required to describe respective processes in a suitable way.

Altogether, more than 100 different variants of the time patterns were identified, which then were consolidated in a set of 10 representative time patterns. Respective time patterns allow us to systematically elicit and analyze the requirements described by process specifications. Moreover, they enable us to analyze and compare the expressiveness of process modeling languages and tools regarding their support of time-aware processes. Revisiting Example 3.1, for example, we can now classify the temporal constraints encountered based on the time patterns:

Example 3.2 (Treatment Process Revisited)
Observing the temporal constraints discussed in Example 1.1 (cf. Fig. 3.1), it can be recapped that the *appointment* for perform treatment constitutes a *Fixed Date Element* (TP4) restricting the earliest start date of the respective activity. Furthermore, its value will be set during run time by activity make appointment. Restricting activity prepare patient to be performed *exactly 1 day before* perform treatment is a *Time Lag between two Activities* from the start of prepare patient to the start of perform treatment (TP1). The constraint regarding the execution time of prepare patient, in turn, represents a *Schedule Restricted Element* (TP5) and the one concerning prepare treatment a maximum *Duration* (TP2). Finally, the treatment plan for perform aftercare constitutes a *Periodicity* (TP10); to be more precise, it constitutes the underlying periodicity rule of the *periodicity* represented by activity perform aftercare.

Systematic Literature Review

Systematic Literature Review

To further assess the relevance and completeness of the identified time patterns as well as to evaluate any possible bias caused by the data sources, a *systematic literature review* [63]

Research Group	Patterns									
	TP1	TP2	TP3	TP4	TP5	TP6	TP7	TP8	TP9	TP10
Bettini et al. (e.g., [11, 12])	X	X	X	X						
Combi et al. (e.g., [24, 27])	X	X	X	X	X	X	X		X	X
Eder et al. (e.g., [43, 45])	X	X		X	X					
Li et al. (e.g., [91, 92])	X	X		X						
Mans et al. (e.g., [98, 99])	X	X		X	X			X		
Marjanovic et al. (e.g., [100, 101])	X	X		X						
Müller et al. (e.g., [103, 104])		X		X						
Sadiq et al. (e.g., [142, 143])	X	X		X					X	
Zhuge et al. (e.g., [174, 175])	X	X		X					X	

Table 3.2: Consolidated results of the systematic literature review (LWR14)

of the primary studies dealing with temporal constraints in business process management was conducted. To the best of our knowledge this has been the first time a systematic effort was made to identify, review, and synthesize the literature on this topic.

The main search term was "temporal constraints business process", but synonyms like "time", "restriction", and "workflow" and any combinations thereof where considered as well. The search strategy identified 1,000 papers of which—after a manual filtering process—73 were identified as primary papers relevant in the context of this work. Having identified relevant publications, each paper is systematically checked for the temporal constraints mentioned.

As the most important result, the analysis does not reveal any temporal constraint relevant for the *control-flow perspective*, which is not covered by the described time patterns. Albeit, the analysis reveals additional temporal constraints that might be relevant for other process perspectives (e. g., validity periods for process data). However, these are out of scope of this thesis.

Consolidated results of the analysis are presented in Table 3.2. In particular, the table only shows the results of the most active groups, where the results taken from the publications of the same group are aggregated in a single row. The analysis shows that all identified time patterns can be found in literature as well. It further shows that certain patterns received more attention in the past than others. Finally, Table 3.2 shows that the time patterns provide the most complete framework regarding temporal constraints in PAIS.

Evaluating Existing Approaches

In a final step, existing approaches and tools were evaluated regarding their support of the time patterns. The evaluation not only considers process management systems and languages, but also includes calendar systems and project management tools, in which temporal constraints play an important role. The evaluation reveals that current support for the time patterns is rather limited and that none of the evaluated systems provides a holistic and integrated support of time-aware processes (see (LWR14) for details).

3.1.3 Evaluation and Related Work

Patterns were first used by Alexander et al. [1] in the 1970s to describe best practices and solutions to recurring problems in architectural design. Alexander defines a pattern to be a reusable solution to a commonly occurring problem. Moreover, the authors note that "[. . .] each pattern represents our current best guess as to what arrangement [. . .] will work to solve the problem presented. The empirical questions center on the problem—does it occur and is it felt in the way we have described it?—and the solution—does the arrangement we propose in fact resolve the problem?" [1]. While the time patterns mainly focus on describing the problem, they also present a first step towards developing a solution.

Patterns also have a long-standing tradition in Computer Science and Software Engineering. For example, Gamma et al. [50] describe 23 design patterns for software engineering that represent tried and tested solutions to commonly occurring problems. Moreover, Buschmann et al. [15–17], Schmidt et al. [147], and Kircher and Jain [62] describe patterns for software architectures and design.

In the field of business process management, workflow patterns have been introduced for different process perspectives to facilitate the comparison of PAIS-enabling technology and to provide a common terminology for existing solutions [137]. Existing workflow patterns, for example, cover process perspectives like control flow [158], data flow [139], resources [140], activities [152], and exceptions [141]. Moreover, patterns for describing control flow changes [167], user interface generation [66], and service interactions [7] were introduced. Respective patterns provide a means for analyzing the expressiveness of process modeling languages and may be used for eliciting and analyzing requirements described by formal process specifications. The introduction of the workflow patterns has had a significant impact on PAIS design as well as on the evaluation of PAISs and process languages [151].

(LWR10) and (LR14) follow in this wake by describing 10 patterns regarding the process time perspective. They extend existing workflow patterns by describing time-related concepts commonly found in business processes and providing a reference system for them. In particular, no such framework for systematically evaluating PAIS with respect to their ability to deal with the time perspective or for eliciting time-related requirements established by process specifications existed in the past.

Most academic approaches dealing with the time perspective of PAISs focus on specific time support features like escalation management [159], scheduling support [6, 25], process monitoring [146], process mining of temporal aspects [161], and verification of temporal constraints [11, 27, 29, 45, 175] (cf. Section 3.2.3 for a more detailed discussion). For example, Combi et al. [27, 29] provide a conceptual model for temporal workflows (i. e., time-aware process schemas), by extending the most common process modeling elements with additional attributes for representing temporal properties (i. e., *Activity Durations*; TP2). Moreover, the authors provide modeling elements for specifying relative temporal constraints (i. e., *Time Lags between Activities*; TP1) and absolute constraints (i. e., *Fixed Date Elements* and *Schedule Restricted Elements*; TP4 and TP5). Based on this model, [27, 29] proposes a technique for checking the temporal consistency of time-aware process schemas. In turn, Eder et al. [45] use Timed Workflow Graphs—an extension of the Critical Path Method (CPM) [149]—to represent temporal properties of activities (i. e., *Activity Durations* and *Fixed Date Elements*; TP2 and TP4) and their control flow relations (i. e., *Time Lags between Activities*; TP1). Based on Timed Workflow Graphs the authors present an approach for calculating activity deadlines in a way such that all temporal constraints are satisfied and the overall process deadline can be met. Marjanovic et al. [100, 101] define a conceptual model for temporal constraints on process schemas. They consider time patterns *Time Lags between Activities* (TP1), *activity* and *process Durations* (TP2), and *Fixed Date Elements* (TP4). Finally, a set of rules for verifying time-aware process schemas is presented.

Overall, the systematic literature review (cf. Table 3.2) has revealed that the time patterns introduced by (LWR14) have been considered in literature before. However, it has further revealed that they provide the most complete framework regarding time support in PAIS. Moreover, related works do not provide a systematic elicitation and elaboration of the requirements for supporting the temporal perspective in PAIS. In particular, in all these works the considered temporal constraints are simply presented as a matter of fact, i. e., there exists no systematic analysis of real-world business processes regarding their actual requirements with respect to the time perspective.

Allen's interval algebra [2] defines the 13 possible relations between two time intervals (e. g., t during s). Allen uses these relationships to propose a system for reasoning about temporal intervals in a hierarchical manner using constraint propagation techniques.

Following their initial publication, the time patterns have already been picked up by several other research groups in different context. For example, Sanchez et al. [145] propose an extended catalog of time patterns. When having a closer look on this work, however, it turns out that they merely propose to consider different variants of the time patterns presented in (LWR14) as separate patterns. In particular, they state that "[the] catalog of Lanz et al. considers such decompositions simple *design choices*; however, we believe that they have such an important impact that the constraints 'Deadline Limit: Lower, Static' and 'Deadline Limit: Excluded, Dynamic' should be considered two different patterns" [145]. We do not fully agree with this conclusion. Moreover, from our understanding the two mentioned pattern variants already constitute two different patterns (i. e., *Fixed Date Element* vs. *Time-based Restrictions*). Though only three of the time patterns proposed by Sanchez et al. are discussed in detail, they seem to be similar to the ones we presented in (LWR14) earlier. Nevertheless, the paper provides an interesting view on the time patterns. In turn, Barba et al. [6] applies the time pattern in the context of scheduling support for declarative workflows. Döhring and Zimmermann [40] apply the time patterns in combination with workflow adaptation patterns. In [22], Time Stream Petri Net (TSPN) representations are proposed for selected time patterns to demonstrate that TSPN constitutes a suitable formalism for supporting the life cycle of processes with temporal constraints. Finally, Lenhard et al. [88, 89] developed a framework for evaluating the degree of support a language provides for different kinds of patterns (including the time patterns). Altogether these works reconfirm that the proposed time patterns are highly relevant in practice.

3.1.4 Discussion and Outlook

In (LWR10) and (LWR14) a10 time patterns are introduced, each of them representing temporal aspects commonly occurring in practice and, hence, being required for the comprehensive support of time-aware processes in PAISs. The analysis of a large set of data sources as well as the systematic literature review conducted have confirmed that the proposed time patterns are relevant in practice and commonly required in various application domains. In combination with existing workflow patterns, the time patterns enable PAIS engineers to choose the process management technology meeting their requirements best. Moreover, the time patterns may be used as a benchmark for assessing the degree of support a particular process management technology provides with respect to the time perspective. In this context, our evaluation of selected approaches and systems has shown that the support of the *time perspective* has been very limited in existing PAISs so far. Similar to workflow patterns, we expect vendors to evaluate their PAISs with respect to their support of the time patterns as well as to extend them towards a better support of time-aware processes.

Alexander et al. [1] defined a pattern to be a reusable solution to a commonly occurring problem. Along this line most pattern catalogs found in literature (cf. Section 3.1.3) focus on first describing the general problem found in practice and then presenting a universal solution to it. By comparison, the time patterns (as well as other workflow patterns [152, 167]) are restricted to the description of a general problem commonly found in practice, but do not provide a readily available solution. In particular, for several time patterns (e. g., *TP6: Time-based Restrictions*, *TP10: Periodicity*), no generally applicable solution is known to date. Moreover, available solutions often significantly depend on their scope of application (e. g., process modeling language, process modeling paradigm). Note that this neither limits the usefulness nor the relevance of the time patterns. In particular, the time patterns are still essential for the comprehensive elicitation of the requirements imposed by real-world business processes as well as for the profound comparison of PAISs with respect to their ability to deal with the time perspective of a business process. Moreover, the time patterns serve as a well-founded and generic basis for the comprehensive integration of the time perspective in future PAIS technology.

Since their introduction, the time patterns have received widespread attention [6, 22, 40, 47, 69, 88, 89, 118, 135, 145, 148, 170]. We expect that the latter will even increase in future works, and a more widespread use in science and industry will be observed. This will be fostered by the definition of a formal semantics for the time patterns as provided in Section 3.2. Moreover, we expect that time patterns for process perspectives other than control flow will emerge (e. g., data, organizational, or resource perspective). The latter are out of the scope of this thesis which focuses on the control flow as the enabling perspective of PAISs.

Altogether, the time patterns present an excellent starting point for developing advanced time-support features, including the verification of temporal constraints in time-aware processes (cf. Section 4.2), escalation management, and advanced scheduling support. Hence, they will significantly contribute to the wider support of time-aware process in PAISs.

3.2 Formal Foundation of Process Time Patterns

The content of this section was published as follows:

(LRW16) A. Lanz, M. Reichert, and B. Weber. Process time patterns: A formal foundation. *Information Systems*, 57:28–68, 2016. DOI: 10.1016/j.is.2015.10.002

The original article is contained in Appendix A.2.

Preliminary results as well as additional details on selected aspects have also been published as a technical report [80].

3.2.1 Problem Description

Our evaluation of existing approaches and tools with respect to their support of the time patterns (cf. Section 3.1.2) revealed ambiguities regarding the semantics of the time patterns as described by their informal descriptions. If respective issues are not resolved through the provision of a precise semantics, time patterns might be interpreted differently. This would hamper both pattern implementation and pattern-based comparison of existing PAIS implementations.

Precise Formal Semantics

To enable a robust and error-free execution of time-aware processes, it becomes necessary to verify the consistency of the time perspective at both design and run time. As a prerequisite for verifying the time perspective of a process schema, a *precise and formal semantics* needs to be provided for each time pattern. In particular, time-aware processes may contain temporal inconsistencies caused by complex interactions among different time pattern occurrences, i. e., due to complex interdependencies a process may not be executable without violating at least one of its temporal constraints. Example 3.3 illustrates some of the processes behind such interactions (adopted from LRW16).

Example 3.3 (Interactions among temporal constraints)
Figure 3.3 depicts a process schema consisting of three activities and two gateways. Each activity is associated with a minimum and maximum duration (Pattern *TP2: Duration*). Furthermore, time lags exist between the end of activity A_1 and the start of activity A_3, between the end of A_1 and the start of A_4, and between the end of A_3 and the end of A_4 (Pattern *TP1: Time Lags between Activities*).

At first glance, the process schema seems to be sound. However, when having a closer look, one realizes that the process schema can never be executed without violating at least one of its temporal constraints. In particular, A_3 may be started the earliest 20 time units after completing A_1 and takes at least 30 time units to complete, i. e., A_3 completes at least 50 time units after completing A_1. In turn, A_4 must start the latest 25 time units after completing A_1 and takes at most 10 time units to complete. Thus, A_4

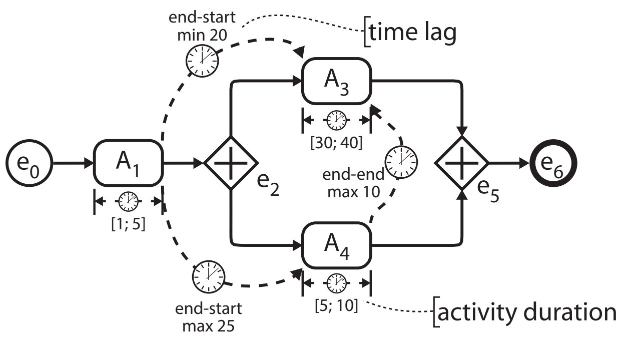


Figure 3.3: Interdependencies among temporal constraints (LRW16)

completes at most 35 time units after completing A_1. However, this violates the time lag between A_3 and A_4. Particularly, it is not possible to complete A_3 within 10 time units after completing A_4.

Example 3.3 demonstrates that the use of a time pattern in the context of a process schema can never be treated in isolation as complex interactions of different time pattern occurrences may result in *hidden effects*. Such complex interactions, in turn, make a formal specification of the semantics of the time patterns almost indispensable. Only then it becomes possible to develop algorithms for detecting inconsistencies. In turn, only through the development of such algorithms a robust and error-free execution of time-aware processes becomes possible.

Any formal semantics for the time patterns needs to specify how the various time patterns interact with the elements of the *control-flow perspective*. At the same time, as a pattern is defined as a reusable solution to a commonly occurring problem, the time patterns should be applicable to a wide range of application scenarios. Therefore, a formal pattern description should be independent of a specific process modeling language or paradigm. Only then time patterns as well as their formal semantics will be widely accepted. Moreover, this constitutes a requirement to enable PAISs engineers to integrate the time patterns into a PAIS without need to cope with language-specific issues of two different languages.

3.2.2 Scientific Contribution

To tackle the issues outlined above, (LRW16) complements previous work on time patterns (cf. Section 3.1) by providing a *precise formal semantics* for each pattern. In particular, these formal semantics contribute to overcome the discussed problems. Moreover, they foster the integration of the time perspective into PAISs.

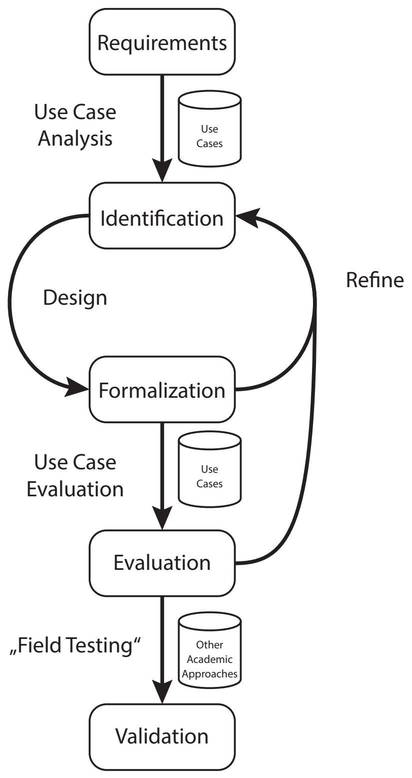


Figure 3.4: Research methodology applied in (LRW16)

To ground the formal semantics of the time patterns on a solid basis, we apply a *design science* approach [53] (cf. Figure 3.4). In a first step, the requirements for specifying formal pattern semantics are determined. In particular, formal pattern semantics

(R1) should be as generic as possible, while at the same time being as specific as required to avoid ambiguities,
(R2) must consider all pattern variants,
(R3) needs to precisely define the effects, the respective pattern has on process enactment,
(R4) needs to consider the impact process instance data may have on an occurrence of a pattern, and
(R5) should be described independent of a particular process modeling language or paradigm (e. g., imperative vs. declarative).

To identify the semantics of each time pattern a large set of real cases, process schemas and process descriptions consisting of more than 430 processes is analyzed (cf. LRW16). Subsequently, for each time pattern the semantics collected from its occurrences in the data sources are merged to derive the overall semantics of the time pattern.

Formal Semantics of Time Patterns

Temporal Execution Trace

To cope with requirements R1, R3, and R5, *temporal execution traces* (*traces* for short) are used as basis for formalizing pattern semantics. In particular, a (temporal) execution trace [162] represents one possible execution of a process schema in terms of all events that occurred during the execution of the respective process instance together with their time stamps (cf. Section 2.2). Thus, we obtain an approach being independent of a particular process modeling language and paradigm; yet, it is closely related to the process execution semantics. By associating the defined pattern semantics with a process meta model, the effects a pattern has on process enactment can be derived in a precise and formal way. Moreover, note that execution traces are a common formalism for describing and analyzing process schemas and corresponding instances (e. g., [95, 111, 134]).

Formally speaking, a temporal execution trace is defined as follows (LRW16):[2]

Definition 3.1 (Temporal Execution Trace)
Let $\mathcal{PS}$ be the set of all process schemas. Let further $\mathcal{E}$ be the set of all *events* and $\mathcal{C}$ be the total set of absolute time points.

Then: Let $\mathcal{E}_S \subseteq \mathcal{E}$ be the set of all *events* that may occur during the execution of an instance I of process schema $S \in \mathcal{PS}$.

The *occurrence* of event $e \in \mathcal{E}_S$ at time point $t \in \mathcal{C}$ is denoted by

$$\varphi = (e, t) \in \mathcal{E}_S \times \mathcal{C}$$

Moreover,

$$\Phi_S \subseteq \mathcal{E}_S \times \mathcal{C}$$

denotes the set of all possible *event occurrences* during the execution of process schema S.

Further: $\mathcal{Q}_S$ denotes the set of all *temporal execution traces* producible on S. A temporal execution trace $\sigma_S \in \mathcal{Q}_S$ is defined as ordered set of event occurrences φ_i:

$$\sigma_S = \langle \varphi_1, \ldots, \varphi_n \rangle, \varphi_i \in \Phi_S, i = 1, \ldots, n, n \in \mathbb{N}$$

Finally, function $occur : \mathcal{Q}_S \times \mathcal{E}_S \mapsto 2^{\Phi_S}$ returns all occurrences of event $e \in \mathcal{E}_S$ in trace $\sigma_S \in \mathcal{Q}_S$.

For the sake of brevity, we use notions φ^e and φ^t when referring to event e or time stamp t of an event occurrence $\varphi = (e, t)$. Moreover, we may omit subscript S if it becomes clear from the context.

[2]Note that to avoid unnecessary repetition this synopsis only provides an abridged description of respective definitions. For more details please refer to the original article.

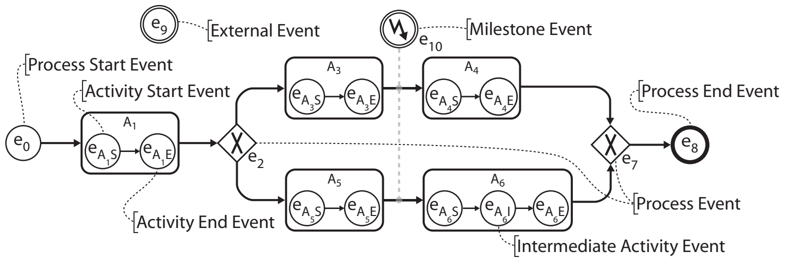


Figure 3.5: Events occurring during a process

In a temporal execution trace, each *activity instance* is represented by the occurrence of at least two events, i. e., the *start* and *end event* of the activity (cf. Figure 3.5). Moreover, intermediate events may be triggered during the execution of an activity instance. In turn, a gateway (e. g., XOR-split or AND-join) is represented by a single event, as—in general—gateways merely serve structuring purposes and, therefore, do not consume any time during process execution. If a gateway does consume time, without loss of generality, this may be represented by an activity directly preceding or succeeding the gateway. Finally, there may be events not bound to an activity or gateway (e. g., external events).

As a particular challenge, for certain time patterns the iteration of a loop structure needs to be taken into account when defining pattern semantics, i. e., it must be possible to determine to which iteration a particular activity instance (or more generally an event occurrence) belongs. In the context of nested loops, it is not sufficient to solely consider the repetition count of the activity itself. Rather, it becomes necessary to always consider the iteration of the inner-most loop with respect to the iteration of any surrounding loop. Consequently, the notion of (loop) iteration is defined as follows (LRW16):

Definition 3.2 (Iteration)
Let $S \in \mathcal{PS}$ be a process schema. Then: A loop L is a process fragment (i. e., $L \subseteq \mathcal{E}_S$) with a Loop-Start as entry node and a corresponding Loop-End as exit node. The set of possible iterations of S is given by $\mathcal{I}_S \subseteq 2^{(2^{\mathcal{E}_S})\times\mathbb{N}}$. Accordingly, the iteration of a loop is defined as ordered set

$$I = \langle (L_0 : n_{L_0}), \ldots, (L_k : n_{L_k}) \rangle \in \mathcal{I}_S.$$

I uniquely identifies each loop and its current iteration with respect to the iteration of any surrounding loop. Thereby, L_0 is the given process schema and L_i $(1 \leq i \leq k)$ the i-th loop structure. In turn, n_{L_i} $(1 \leq i \leq k)$ designates the iteration count of loop L_i with respect to its directly surrounding loop L_{i-1}.

Function *iter* : $\mathcal{Q}_S \times \Phi_S \mapsto \mathcal{I}_S$ returns the current iteration of the inner-most loop surrounding event φ^e in trace σ_S.

To illustrate the concepts of temporal execution trace and loop iteration consider Example 3.4.

Example 3.4 (Temporal Execution Trace and Iteration)
Consider the process schema depicted in Figure 3.6. An example of a temporal execution trace on this process schema is as follows:[3]

$$\begin{aligned}\sigma = \langle &(e_0, 2), (e_{\mathsf{A}_1 S}, 3), (e_{\mathsf{A}_1 E}, 5), (e_2, 6), (e_{\mathsf{A}_3 S}, 9), (e_{\mathsf{A}_3 E}, 12), (e_4, 13), (e_5, 14),\\ &(e_{\mathsf{A}_6 S}, 19), (e_{\mathsf{A}_6 E}, 22), (e_8, 23), (e_9, 24), , (e_{\mathsf{A}_{10} S}, 27), (e_{\mathsf{A}_{10} E}, 29), (e_{11}, 30),\\ &(e_{\mathsf{A}_{12} S}, 35), (e_{\mathsf{A}_{12} E}, 37), (e_{13}, 38)\rangle\end{aligned}$$

This trace indicates that the corresponding process instance was started at time 2 (i. e., $(e_0, 2)$). At time 3, activity A_1 was started (i. e., $(e_{\mathsf{A}_1 S}, 3)$). In turn, A_1 was completed at time 5 (i. e., $(e_{\mathsf{A}_1 E}, 5)$); i. e., A_1 had a duration of $5 - 3 = 2$. Next, event e_2 occurred at time 6 followed by the execution of activity A_3 from time 9 to 12. After the execution of e_4 and e_5 at time 13 and 14, respectively, the upper path was chosen, which is indicated by the occurrence of start event $e_{\mathsf{A}_6 S}$ related to A_6 at time 19. Note that none of the events of the lower path occurs within this trace as the respective path has been skipped. After completing A_6 events e_8, e_9, $e_{\mathsf{A}_{10} S}$, $e_{\mathsf{A}_{10} E}$, e_{11} and $e_{\mathsf{A}_{12} S}$ are triggered; i. e., neither loop L_2 nor loop L_1 is repeated. Finally after completing A_{12} at time 37, event e_{13} is triggered at time 38 completing the instance.

Another trace (without timestamps) for this process schema may be as follows:

$$\sigma = \underbrace{\langle e_0, e_{\mathsf{A}_1 S}, e_{\mathsf{A}_1 E},}_{\langle(L_0{:}1)\rangle} \underbrace{e_2, e_{\mathsf{A}_3 S}, e_{\mathsf{A}_3 E},}_{\langle(L_0{:}1),(L_1{:}1)\rangle} \underbrace{e_4, e_5, e_{\mathsf{A}_6 S}, e_{\mathsf{A}_6 E}, e_8, e_9,}_{\langle(L_0{:}1),(L_1{:}1),(L_2{:}1)\rangle} \underbrace{e_4, e_5, \boxed{e_{\mathsf{A}_7 S}}, e_{\mathsf{A}_7 E}, e_8, e_9,}_{\langle(L_0{:}1),(L_1{:}1),(L_2{:}2)\rangle} \dots$$

$$\dots, \underbrace{e_{\mathsf{A}_{10} S}, e_{\mathsf{A}_{10} E}, e_{11},}_{\langle(L_0{:}1),(L_1{:}1)\rangle} \underbrace{e_2, e_{\mathsf{A}_3 S}, e_{\mathsf{A}_3 E},}_{\langle(L_0{:}1),(L_1{:}2)\rangle} \underbrace{e_4, e_5, e_{\mathsf{A}_6 S}, e_{\mathsf{A}_6 E}, e_8, e_9,}_{\langle(L_0{:}1),(L_1{:}2),(L_2{:}1)\rangle} \underbrace{e_{\mathsf{A}_{10} S}, e_{\mathsf{A}_{10} E}, e_{11},}_{\langle(L_0{:}1),(L_1{:}2)\rangle} \dots$$

$$\dots, \underbrace{e_{\mathsf{A}_{12} S}, e_{\mathsf{A}_{12} E}, e_{13}\rangle}_{\langle(L_0{:}1)\rangle}$$

In this trace, loop L_2 is repeated twice within the first iteration of L_1 and once within the second iteration of L_1. This can be seen when considering the value of $\mathit{iter}(\sigma, \varphi)$ which is given for each event below the trace. For example, regarding the occurrence of event $e_{\mathsf{A}_7 S}$ during the 2nd iteration of loop L_2 within the 1st iteration of loop L_1, we obtain

$$\mathit{iter}\,(\sigma, (e_{\mathsf{A}_7 S}, \cdot)) = \langle (L_0 : 1), (L_1 : 1), (L_2 : 2)\rangle$$

This is indicated in the last group of the first line of the execution trace.

[3]Without loss of generality, we use integers starting at 0 for representing absolute time points $c \in \mathcal{C}$.

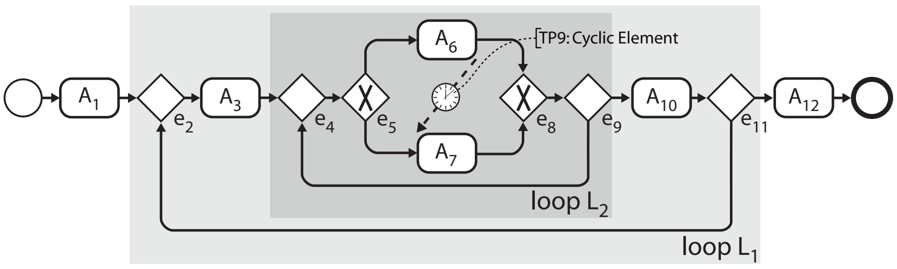


Figure 3.6: Nested loops and iterations

Armed with these tools it becomes possible to formally define the conditions under which a temporal execution trace $\sigma_S \in \mathcal{Q}_S$ is *temporally compliant* with the occurrence of a particular time pattern defined on process schema $S \in \mathcal{PS}$. In turn, a temporal execution trace σ_S is *temporally compliant* with the set of constraints defined on S if and only if it is temporally compliant with each of the corresponding constraints on S.

Temporally Compliant

To avoid excessive overlapping and to reduce repetitions, in the following, only selected time pattern semantics are sketched (see LRW16 for details).

We illustrate the definition of time pattern semantics along time pattern *TP1: Time Lags between Activities*, which allows restricting the time span between the starting/ending instants of two activities A and B (cf. Section 3.1.2). Time pattern TP1 has several *design choices* that need to be covered by respective pattern semantics (cf. LWR14):

Design Choice D: A time lag may express a minimum or maximum time distance, or both.

Design Choice E A time lag may represent a start-to-start, start-to-end, end-to-start, or end-to-end relationship between activities.

Moreover, it is noteworthy that a time lag may not only be specified between directly succeeding activities, but between two arbitrary activities that may be conjointly executed in the context of the same process instance. The latter implies that a time lag is trivially fulfilled if one or both activities are not executed in the context of a particular process instance, i. e., respective events do not occur within the corresponding trace.

In the context of loops, ambiguities regarding the interpretation of a time lag between two activities, where one activity resides inside a loop and the other one outside that loop, may arise. Respective ambiguities have to be resolved by the definition of the pattern semantics. A close examination of respective cases and the considered data sources has shown that only one of the possible interpretations is generally meaningful and required in practice. In particular, a time lag entering a loop (i. e., whose source activity is outside the loop while the target activity is inside that loop) only applies to the first iteration of that loop, whilst a time lag exiting a loop only applies to the last iteration of that loop. In the context of nested iterations this is extended appropriately.

In general, the semantics of time pattern TP1 may be expressed as follows (LRW16):

Definition 3.3 (Pattern Semantics TP1)
Let $\mathcal{C}$ be the total set of absolute time points and $\mathcal{D}$ be the set of relative time distances. Let e_X and e_Y represent the source and target event of the time lag. In particular, depending on the kind of relationship, the time lag represents (i. e., start-to-start, start-to-end, end-to-start, or end-to-end), event e_X either corresponds to the start or end event of the source activity. Likewise, e_Y either corresponds to the start or end event of the target activity.

Then: A temporal execution trace σ is *temporally compliant* with an occurrence of the pattern, iff it satisfies the following condition:

$$\begin{aligned}&\forall \varphi \in \mathit{occur}(\sigma, e_X), \forall \psi \in \mathit{occur}(\sigma, e_Y):\\ &\qquad \text{valid}(\sigma, \varphi, \psi) \Rightarrow \text{compareR}(\varphi^t, \psi^t, \text{distance}(\sigma, e_X, e_Y, \psi^t))\end{aligned} \tag{3.1}$$

Thereby function *occur* returns all occurrences of event e_X/e_Y in trace σ. Moreover, function valid : $\mathcal{Q}_S \times \Phi_S \times \Phi_S \mapsto \mathit{Boolean}$ is used to indicate whether a pair of event occurrences is valid, i. e., whether their respective iterations are consistent with the pattern:

$$\text{valid}(\sigma, \varphi, \psi) \equiv (\,\mathit{iter}(\sigma, \varphi) = \mathit{iter}(\sigma, \psi)) \vee (\,\mathit{iter}(\sigma, \varphi) \parallel \mathit{iter}(\sigma, \psi)) \tag{3.2}$$

This means that the iterations of the two event occurrences are either the same or adjacent. The latter is represented by the adjacency operation $\parallel$, which is true iff either the second operand represents the first iteration of an inner loop of the first operand, or the first operand represents the last iteration of an inner loop of the second operand (cf. LRW16 for a formal definition).

Function distance : $\mathcal{Q}_S \times \mathcal{E}_S \times \mathcal{E}_S \times \mathcal{C} \mapsto [\mathcal{D}]$[4], which is used in Formula 3.1, determines the parameter value of the pattern occurrence identified by events $e_X \in \mathcal{E}_S$ and $e_Y \in \mathcal{E}_S$ as it is effective in trace $\sigma \in \mathcal{Q}_S$ at time $t \in \mathcal{C}$. In particular, the parameter value (i. e., minimum or maximum distance value) of a time lag may change during process enactment. Hence, it becomes necessary to determine which of these values shall be valid for a particular instance of a pattern occurrence.

Finally, whether or not a pair of event occurrences satisfies a time lag depends on the kind of temporal distance represented by the pattern occurrence (i. e., minimum distance, maximum distance or interval). This is defined by function compareR : $\mathcal{C} \times \mathcal{C} \times [\mathcal{D}] \mapsto$ *Boolean* with[5]

$$\text{compareR}(\varphi^t, \psi^t, d) \equiv \begin{cases} \varphi^t + \min(d) \leq \psi^t & \text{if minimum distance} \\ \psi^t \leq \varphi^t + \max(d) & \text{if maximum distance} \\ \varphi^t + \min(d) \leq \psi^t \leq \varphi^t + \max(d) & \text{if time interval} \end{cases} \tag{3.3}$$

[4]We use notation $[\mathbf{X}]$ to indicate the set of intervals over domain X, i. e., $[\mathbf{X}] = \{[x_1, x_2] | x_1, x_2 \in X \wedge x_1 \leq x_2\}$.

[5]Note that $\min(d)/\max(d)$ represents the minimum / maximum value of interval d, i. e., $\min(d) = \min\{x | x \in d\}$ and $\max(d) = \max\{x | x \in d\}$.

Example 3.5 (Pattern Semantics TP1)
We illustrate the semantics of TP1 along process schema S from Figure 3.7 (LWR14). S contains a maximum time lag of 20 between the start of A_4 and the start of A_7. Moreover, it contains a minimum time lag between the end of A_1 and the start of A_6. The parameter value of this time lag is consumed from data element `time lag`, which, in turn, is first produced by A_1 and may later be modified by A_4. Based on process schema S, for example, traces σ_1 and σ_2 can be produced.

$$\begin{aligned}\sigma_1 = \Big\langle &(e_0, 0), (e_{\mathsf{A}_1 S}, 1), (e_{\mathsf{A}_1 E}, 3)_{\{\text{distance}(\sigma, e_{\mathsf{A}_1 E}, e_{\mathsf{A}_6 S}, t \geq 3) \leftarrow [10,\infty]\}}, (e_2, 4),\\ &(e_{\mathsf{A}_3 S}, 10), (e_{\mathsf{A}_3 E}, 12), (e_5, 13), (e_{\mathsf{A}_6 S}, 14), (e_{\mathsf{A}_6 E}, 15),\\ &(e_{\mathsf{A}_7 S}, 17), (e_{\mathsf{A}_7 E}, 19), (e_8, 20)\rangle\end{aligned}$$

$$\begin{aligned}\sigma_2 = \Big\langle &(e_0, 0), (e_{\mathsf{A}_1 S}, 2), (e_{\mathsf{A}_1 E}, 5)_{\{\text{distance}(\sigma, e_{\mathsf{A}_1 E}, e_{\mathsf{A}_6 S}, t \geq 2) \leftarrow [8,\infty]\}}, (e_2, 6),\\ &(e_{\mathsf{A}_4 S}, 8), (e_{\mathsf{A}_4 E}, 10)_{\{\text{distance}(\sigma, e_{\mathsf{A}_1 E}, e_{\mathsf{A}_6 S}, t \geq 10) \leftarrow [16,\infty]\}}, (e_5, 11),\\ &(e_{\mathsf{A}_6 S}, 13), (e_{\mathsf{A}_6 E}, 19), (e_{\mathsf{A}_7 S}, 25), (e_{\mathsf{A}_7 E}, 27), (e_8, 28)\rangle\end{aligned}$$

In this context, $(e_{\mathsf{A}_1 E}, 3)_{\{\text{distance}(\sigma, e_{\mathsf{A}_1 E}, e_{\mathsf{A}_6 S}, t \geq 3) \leftarrow [10,\infty]\}}$ indicates that after the occurrence of event $e_{\mathsf{A}_1 E}$ the value of $\text{distance}(\sigma, e_{\mathsf{A}_1 E}, e_{\mathsf{A}_6 S}, \cdot)$ is changed to $[10, \infty]$, i. e., $\forall\, t \geq 3 : \text{distance}(\sigma, e_{\mathsf{A}_1 E}, e_{\mathsf{A}_6 S}, t) = [10, \infty]$ (cf. trace σ_1).

Trace σ_1 is temporally compliant with the set of temporal constraints defined on S. In particular, the time lag between A_1 and A_6 is set to 10 by A_1. Accordingly, it is satisfied as

$$\begin{aligned}&\text{compareR}(\varphi^t_{e_{\mathsf{A}_1 E}}, \varphi^t_{e_{\mathsf{A}_6 S}}, \text{distance}(\sigma, e_{\mathsf{A}_1 S}, e_{\mathsf{A}_6 S}, \varphi^t_{e_{\mathsf{A}_6 S}})) \equiv\\ &\qquad \varphi^t_{e_{\mathsf{A}_1 E}} + \min(\text{distance}(\sigma, e_{\mathsf{A}_1 S}, e_{\mathsf{A}_6 S}, 14)) = 3 + 10 = 13 \leq \varphi^t_{e_{\mathsf{A}_6 S}} = 14\end{aligned}$$

holds. Further the time lag between A_4 and A_7 is trivially fulfilled, as A_4 is not executed.

In turn, trace σ_2 is not temporally compliant with the temporal constraints on S. On one hand the time lag between A_4 and A_7 is satisfied:

$$\varphi^t_{e_{\mathsf{A}_4 S}} + \max(\text{distance}(\sigma, e_{\mathsf{A}_4 S}, e_{\mathsf{A}_7 S}, 25)) = 8 + 20 = 28 \geq \varphi^t_{e_{\mathsf{A}_7 S}} = 25$$

On the other, the time lag between A_1 and A_6 is violated. In particular, the parameter value of the time lag is set to 8 by A_1. However, later it is updated to 16 by A_4. Thus, it holds,

$$\varphi^t_{e_{\mathsf{A}_1 E}} + \max(\text{distance}(\sigma, e_{\mathsf{A}_1 E}, e_{\mathsf{A}_6 S}, 13)) = 5 + 16 = 21 \not\leq \varphi^t_{e_{\mathsf{A}_6 S}} = 13$$

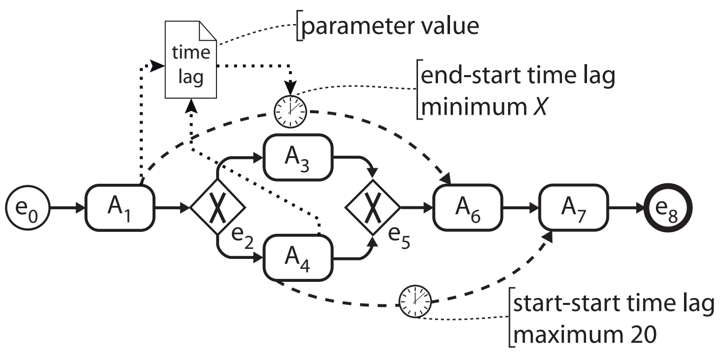


Figure 3.7: Illustration of pattern semantics TP1

Note that the semantics of time patterns *TP2: Durations*, *TP3: Time Lags between Arbitrary Events*, and *TP9: Cyclic Elements* can be defined similarly (using different functions for valid and distance) as they all restrict the relative time distance between pairs of events (cf. Pattern Semantics 2–4 in LRW16).

In turn, patterns *TP4: Fixed Date Elements* and *TP7: Validity Period* refer to an absolute point in time. Hence, their formal semantics can be defined by comparing the time of the occurrence of the respective event with the parameter value of the corresponding pattern occurrence being effective at that time. Formally, this can be expressed by the following condition:

$$\forall \varphi \in \mathit{occur}(\sigma, e) : \mathrm{compareA}(\varphi^t, \mathrm{date}(\sigma, e, \varphi^t)) \tag{3.4}$$

Depending on the design choices selected for the pattern occurrence

(i) e corresponds to the start or end event of the respective process element (i. e., activity or process),
(ii) $\mathrm{compareA} : \mathcal{C} \times \mathcal{C} \rightarrow \mathit{Boolean}$ represents the kind of restriction (i. e., earliest/latest start; earliest/latest completion), and
(iii) $\mathrm{date} : \mathcal{Q}_S \times \mathcal{E}_S \times \mathcal{C} \rightarrow \mathcal{C}$ represents the parameter value of the pattern occurrence being effective at time φ^t.

Note that for *TP7* the value of date is independent of both the process instance σ and the time (i. e., $\mathrm{date}(\cdot, e, \cdot) \equiv \mathrm{const}$), while for *TP4* this does not apply.

A third group of time patterns is based on repetitive time slots (e. g., based on expressions like *Mo–Fr, 8 am–5 pm*). To avoid any restrictions regarding the representation of such a schedule, we only require that it can be materialized as a set of intervals on the absolute time points $\mathcal{C}$; i. e., a *schedule s* is defined to be a possibly infinite set of continuous intervals on $\mathcal{C}$: *Schedule*

$$s \subset [\mathcal{C}] = \{[t_{min}, t_{max}] | t_{min}, t_{max} \in \mathcal{C} \wedge t_{min} \leq t_{max}\}$$

Based on this abstract representation, it becomes possible to define the semantics of time patterns *TP5: Schedule Restricted Elements* and *TP6: Time-based Restrictions* (cf.

Pattern Semantics 7 and 8 in LRW16). Regarding the semantics of TP6, we observed a subtle difference between the different pattern variants found in the data sources, which has not been covered by the original design choices. In order to completely cover the semantics of TP6, therefore, we added a *design choice* that allows specifying how the execution time frames of the activities (processes) referred to by the respective pattern occurrence and the reference time frame shall be compared (cf. LRW16).

Finally, the semantics of *TP8: Time-dependent Variability* and *TP10: Periodicity* can be defined based on the existing semantics of different workflow patterns (e. g., exclusive choice and deferred choice) and a combination of different time patterns. Note that these time patterns are still useful and required as they provide an abstract view on complex situations. Moreover, as discussed in Section 3.1, in general, the periodicity rule of a *Periodicity* only becomes known during run time. When using a complex combination of time and workflow patterns to specify a pattern occurrence, however, it would be necessary to pre-specify a corresponding process fragment at design time, which is not possible as the periodicity rule is unknown.

Based on the defined pattern semantics it becomes possible to check whether a given temporal execution trace is temporally compliant with all temporal constraints defined on a given process schema. This gives rise to the following definition of temporal consistency of a process schema (cf. Definition 12 in LRW16).

Definition 3.4 (Temporal Consistency)
A process schema $S \in \mathcal{PS}$ is consistent with the set of temporal constraints defined on S, if for each possible *execution path* (i. e., each possible set of nodes that may be executed during a single process instance) there exists at least one temporal execution trace $\sigma \in \mathcal{Q}_S$ being temporally compliant with the set of temporal constraints defined on S.

In particular, it must be ensured that for each possible path of a process schema (i. e., each path that may be selected during process enactment) there exists at least one temporal execution trace being temporally compliant with all temporal constraints of the process schema. Otherwise, the time perspective of the process schema is inconsistent as it contains conflicting temporal constraints, i. e., the process schema contains a path that cannot be executed without violating at least one of the temporal constraints.

3.2.3 Evaluation and Related Work

As mentioned in Section 3.1.3, considerable work related to the verification of temporal constraints in processes exists. Respective approaches come along with an either implicit or explicit semantics of the time pattern variants they support. In particular, in most cases respective semantics may be derived from the formalism used for verifying the temporal constraints (e. g., temporal constraint networks [39] or project network techniques [112]) and the proposed translation procedure from the process schema to the respective formalism.

Both Bettini et al. [11] and Combi et al. [27, 28] use Simple Temporal Network (STN) [39] as basic formalism for representing temporal constraints in processes and reasoning about them. Simply speaking, an STN is a directed graph whose vertices represent time-point variables (called timepoints) and whose arcs correspond to temporal constraints between these variables (cf. Section 4.1 for a more detailed discussion of STN). A process schema is translated into a set of STNs by decomposing it along its XOR-splits, i. e., each resulting STN corresponds to one possible execution path. In turn, in an STN each node in the time-aware process schema is represented by two timepoints—expressing the starting and ending instant of the node—and temporal constraints in the STN are used to represent precedence relationships (i. e., control edges) and temporal constraints in the process schema [11, 27]. Compared to Bettini et al., Combi et al. [27] additionally consider well-nested loops by assuming that each loop is associated either with a minimum and maximum number of iterations or a maximum duration for completing all loop iterations. This enables them to transform each loop into a nested set of XOR-blocks. In general, the semantics of the time pattern inherent to the approaches by Bettini et al. [11] and Combi et al. [27] is similar to the one presented in (LRW16) (cf. Table 3.3).

The Timed Workflow Graph used by Eder et al. [45] constitutes an extension of the Critical Path Method [112]. Timed Workflow Graphs are similar to process schemas, representing each activity by a single node and control flow dependencies (i. e., control edges) by structural temporal constraints. Additionally, explicit temporal constraints may be used to represent temporal constraints of the process. Note that from an operational point of view both structural temporal constraints and explicit temporal constraints are treated exactly the same. An important difference between the formal semantics presented by (LRW16) and the one inherent to the approach presented in [45], is that the latter assumes that activity durations are deterministic (i. e., the same for all instances). The authors are aware that this represents a limitation of their approach and propose to use a probabilistic approach based on Program Evaluation and Review Technique (PERT) [112] to overcome this limitation [45, 114].

Zhuge et al. [175] do not use any specific formalism. Instead they define temporal constraints in terms of restrictions of the start and completion times of activities. Moreover, they do not presuppose any specific process modeling language or paradigm, but describe process schemas in terms of temporal relationships. Note that this is similar to the way the semantics presented in (LRW16) is defined. However, there exist significant differences, for example, in the way the duration of a process (TP2) is defined (cf. Table 3.3). In particular, the definition by [175] only considers the minimally and maximally possible duration of a process, but factors out the actual duration of a process instance. According to the analyzed the data sources, however, this does not meet the intended semantics of a process duration (i. e., TP2 variant C[c]), i. e., we are convinced that the provided definition of the respective pattern semantics meets the intended semantics of this pattern variant best.

Pattern		Bettini et al.[ab]	Combi et al.[b]	Eder et al.[ab]	Zhuge et al.[ab]
TP1		(+)	(+)	(+)/(∗)	(+)/○
TP2	C[a]	(+)	(+)	(∗)/○	(+)
	C[c]	(∗)	(∗)	○	–
TP3		(∗)	(∗)	○	○
TP4	C[a]	(∗)	(+)/○	(∗)/○	(+)
	C[c]	○	○	○	○
TP5	C[a]	?	?	(+)/(∗)	○
	C[c]	○	○	○	○
TP6		○	○	○	○
TP7	C[a]	(∗)	(∗)/○	(∗)/○	(∗)
	C[c]	○	○	○	○
TP8		○	○	○	○
TP9		○	(∗)/○	○	○
TP10		○	?	○	○

(+)	Equivalent to a restricted variant of the pattern semantics.
(∗)	Not discussed/No implementation provided, but may be implemented with a semantics equivalent to a restricted variant of the presented pattern semantics.
?	Discussed but no implementation is provided.
–	Different semantics (cf. [80] for details).
○	Not considered.

[a] Does not consider loops.

[b] Does not consider dynamic changes of the parameter value of a pattern occurrence during run time.

Table 3.3: Assessment of Different Approaches [80]

Table 3.3 provides an overview of related approaches, the time patterns considered by them and the similarity of respective pattern semantics compared to the ones defined in (LRW16). In particular, note that only a small subset of the time patterns and their variants is actually supported by the considered approaches. The latter were selected based on the systematic literature review we conducted as part of (LWR14), as they provide the broadest support of the time patterns as well as at least some kind of semi-formal description of the temporal constraints considered. A more detailed analysis of respective approaches and a more detailed discussion of the results summarized by Table 3.3 can be found in [80].

In terms of other workflow patterns, formal semantics have been defined for patterns covering the control flow perspective [115] as well as process changes patterns [134]. Respective semantics are defined based on pi-calculus [115] and execution traces [134]. However, note that time patterns significantly differ from other workflow patterns, as interactions between different time patterns may be complex, resulting in hidden effects (cf. Example 3.3). Hence, for the time patterns the provision of a precise formal semantics is even more important. Moreover, the resulting ability to discover and analyze such effects is indispensable. (LRW16) is the first attempt that provides such precise and formal semantics for all time patterns.

The presented formalization uses temporal execution traces [162] to define the semantics of the time patterns in a precise and formal way. Note that there exists a plethora of other techniques for describing the temporal properties of a system, which could be used as well. Examples include different kinds of temporal logic [4, 51, 52, 59, 68], timed Petri nets [10, 13, 19, 41, 117], and timed automata [3, 37, 97]. We chose temporal execution traces for several reasons. On one hand, they are easy to use, on the other, they closely resemble the execution semantics of processes, while being independent of a particular process modeling paradigm (i. e., declarative vs. imperative process modeling). In particular, an execution trace represents a possible execution of a process schema fully independent of the way the respective schema has been specified. Moreover, most PAISs provide an execution log (cf. Section 2.2) comprising the events (and their time of occurrence) related to the start and completion of activities and indicating process instance they belong to as well as different additional information (e. g., executing agent). Such execution log is similar to a temporal execution trace. Amongst others, this makes it easy to implement techniques for checking conformance [136] of a process instance with respect to a given process schema and its temporal constraints based on the defined formal semantics of the time patterns; i. e., answering the question *do the log and the process schema conform with each other* [136]—a problem statement common in monitoring and auditing of business processes.

3.2.4 Discussion and Outlook

In (LRW16), we have formally defined the semantics of the time patterns originally introduced in (LWR10; LWR14) (cf. Section 3.1). Respective pattern semantics have been identified by means of an extensive analysis of a large set of industrial cases, process schemas, and process descriptions. The formal description of the proposed pattern semantics are expressed independently of a particular process meta model to foster the use of the time patterns for a wide range of process modeling languages as well as to ease pattern implementation in existing PAISs.

The elicitation and development of the formal pattern semantics gave us the opportunity to uncover some aspects that might otherwise have been neglected when using only an informal pattern description; e. g., regarding the relationship between loops and *Time Lags between Activities* (TP1) or the handling of *Time-based Restrictions* (TP6). Moreover, the defined formal semantics enable us to answer open issues, e. g., related to the interactions between the *time* and *control-flow perspective* or the interpretation of certain combinations of process elements. Finally, they constitute a key requirement for developing techniques that allow us to formally verify the time perspective of processes at both design and run time.

Any kind of work trying to formally describe aspects of the real-world always carries some risks regarding its validity. In particular, the main threats we identified with respect to this work include *inaccuracy in specifying pattern semantics*, the *identification procedure* of the pattern semantics, and *completeness* of the patterns and their semantics. (LRW16) analyzes each of these potential threats and describes how we dealt with them. Nevertheless, we do not claim to provide the proper semantics for all possible cases. For example, certain time patterns might have to be interpreted stricter in some scenarios, whereas they can be considered less restrictive in others. Moreover, the proposed time pattern semantics constitutes just one possible way of describing the semantics of the time patterns; i. e., there may be others being appropriate as well. However, our analysis of other approaches, which implicitly provide a semantics for some of the time patterns [80], has revealed that respective semantics are similar to the one defined by (LRW16).

A limitation of the presented formal semantics and an avenue for future work arises from the fact that the formal semantics neither consider design- nor run-time support of time-aware processes. At design time proper support for designing error-free process schemas must be provided; i. e., it must be possible to verify the consistency of a time-aware process schema and to guide PAIS engineers in the design of a consistent time-aware process schema. In turn, when executing a time-aware process schema, challenging issues emerge like "How can temporal constraints be monitored and—if necessary—be enforced?" and "What happens if a temporal constraint can no longer be satisfied?". Furthermore, in certain scenarios, activity *durations* need to be restricted to ensure that a process instance can be completed without violating any constraint, whereas in other scenarios it must be ensured that the entire range of the activity *duration* is available for executing the activity. Some of these challenges are picked up in the following chapters.

4 Managing Time-Aware Processes

You may delay, but time will not.

(Benjamin Franklin, 1706–1790)

4.1 Temporal Consistency of Time-Aware Processes

The content of this section was published as follows:

(LRW16) A. Lanz, M. Reichert, and B. Weber. Process time patterns: A formal foundation. *Information Systems*, 57:28–68, 2016. DOI: 10.1016/j.is.2015.10.002

The original articles is added in Appendix A.2.

4.1.1 Problem Description

An important aspect of any PAIS is its ability to execute a process schema in a robust way. As discussed in Section 1, a prerequisite for robust and error-free process execution is the soundness of the underlying process schema. In the context of *time-aware processes*, this presupposes the *temporal consistency* of the process schemas as well.

Verifying consistency of the time perspective of processes is particularly challenging as temporal inconsistencies may be caused due to complex interactions among different temporal constraints (cf. Example 3.3). The latter significantly differentiates the time

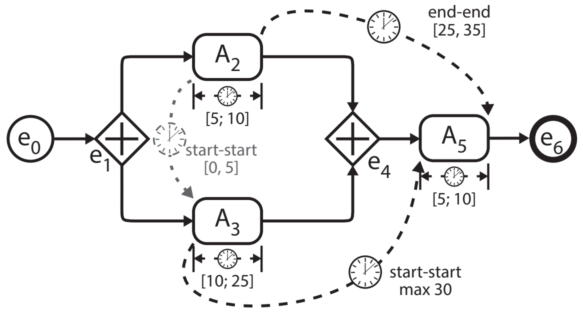


Figure 4.1: Implicit temporal constraints of a time-aware process schema

perspective from other process perspectives. Moreover, respective interactions might result in hidden effects and implicit temporal constraints which need to be obeyed in order to guarantee successful execution of respective process instances. Example 4.1 illustrates some of the processes behind such interactions and the impact implicit temporal constraints might have on process execution.

Example 4.1 (Implicit temporal constraints)
Figure 4.1 depicts a process schema consisting of three activities and two gateways. Each activity is associated with a minimum and maximum duration. Furthermore, time lags exist between the end of activity A_2 and the end of A_5 as well as between the start of A_3 and the start of A_5.

At first glance, it seems that the execution of activities A_2 and A_3 may take place completely independent of each other as there exists no direct (temporal) relationship between the two. However, when taking a closer look, one realizes that in order to be able to simultaneously satisfy the time lag between A_2 and A_5, the duration constraint for activity A_5 as well as the time lag between A_3 and A_5, it becomes necessary that A_3 has to be started 0 to 5 time units after the start of A_2 (as indicated by the dashed time lag between A_2 and A_3 in Figure 4.1).

If implicit temporal constraints are not obeyed during process run time the violation of one of the explicit temporal constraints will occur in the further course of the process instance. Hence, it is important for the PAIS to be aware of such implicit temporal constraints as well as to monitor and ensure their adherence during run time.

4.1.2 Scientific Contribution

This part of the thesis introduces the ATAPIS framework for supporting time-aware processes in adaptive PAISs. In a first step, techniques are provided for verifying the temporal consistency of time-aware process schemas according to the defined pattern semantics.

To check whether a given time-aware process schema is temporally consistent, ATAPIS maps it to a *Conditional Simple Temporal Network* (CSTN) [57, 154].[1] On one hand, CSTN allows us to exploit and reuse provably sound checking algorithms [154] for a well founded model for representing temporal constraints. On the other hand, CSTN is similar to the execution traces used for defining pattern semantics. Moreover, CSTN allows capturing the complex interdependencies between temporal constraints that cannot be properly captured in process schemas. Hence, they provide a suitable basis for implementing the time patterns and their semantics.

Formally a CSTN is defined as follows:

Definition 4.1 (Conditional Simple Temporal Network)
A *Conditional Simple Temporal Network* (CSTN) [57] is a 6-tuple $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$, where:

- $\mathcal{T}$ is a set of real-valued variables, called *timepoints*.
- P is a finite set of *propositional letters* (or propositions).
- $L : \mathcal{T} \to P^*$ is a function assigning a *label* to each timepoint in $\mathcal{T}$; a *label* is any (possibly empty) conjunction of (positive or negative) letters from P. In the following we use small Greek letters $\alpha, \beta, \ldots$ to denote arbitrary labels. The empty label is denoted by $\boxdot$.
- $\mathcal{C}$ is a set of *labeled simple temporal constraints* (*constraint* in the following); each constraint $c_{XY} \in \mathcal{C}$ has the form $c_{XY} = \langle [x, y]_{XY}, \beta \rangle$, where $X, Y \in \mathcal{T}$, $-\infty \leq x \leq y \leq \infty$, and $\beta \in P^*$ is a label.
- $\mathcal{OT} \subseteq \mathcal{T}$ is a set of *observation timepoints*.
- $O : P \to \mathcal{OT}$ is a bijection that associates an observation timepoint to each propositional letter from P.

Finally, a CSTN contains one special timpoint $Z \in \mathcal{T}$ representing time point zero (i. e., $Z = 0$).

Timepoints represent instantaneous events that may be associated with the start/end events of activities. A *constraint* $c_{XY} = \langle [x, y]_{XY}, \beta \rangle$ expresses that the time span between timepoints X and Y must be at least x and at most y, i. e., $x \leq Y - X \leq y$. When reaching an *observation timepoint*, a decision regarding possible execution paths is made. Formally speaking, when executing observation timepoint P, the truth-value of the associated proposition (i. e., $O^{-1}(P)$) is determined. The *label* attached to each timepoint (constraint) indicates possible executions of the CSTN, i. e., a particular

[1]Note that in later work this has been extended to use more sophisticated models of temporal reasoning (cf. Section 4.2).

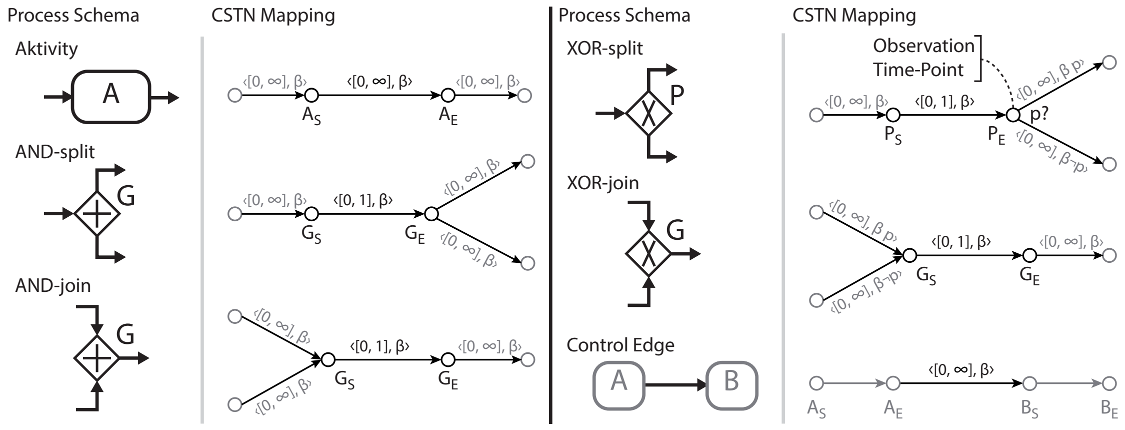


Figure 4.2: Process modeling elements and their mapping to CSTN (LRW16)

timepoint (constraint) will be only considered if its label is satisfiable in the respective instance.

Basically, CSTN timepoints are equivalent to events, whereas a timepoint bound to a specific value is equal to an event occurrence. Moreover, CSTN constraints are specified as inequalities of the form $Y - X \leq t_{max}$ and $t_{min} \leq Y - X$, where X and Y are timepoints and t_{min}/t_{max} correspond to a relative time distance. This is similar to the definition of relative time distance between events as, for example, defined by the semantics of TP1 (cf. function compareR in Formula 3.3). Moreover, an absolute point in time can be represented as the relative time distance between time point zero (i. e., timepoint $Z \in \mathcal{T}$) and the respective *timepoint* $X \in \mathcal{T}$. This mapping results in the same restriction as expressed by function compareA (cf. Formula 3.4). Altogether, it becomes possible to map a time-aware process schema to a CSTN preserving the semantics of the respective time patterns.

When mapping a time-aware process schema to a corresponding CSTN, first of all, the control flow of the process schema is mapped to the CSTN as illustrated in Figure 4.2. Note that each control flow element implicitly represents a temporal constraint, e. g., a control edge is equivalent to a minimum time lag of 0 between its source and target node.

In turn, loop structures cannot be directly mapped to CSTN. Note that at build-time the actual number of iterations, and thus the number of occurrences of corresponding events, is unknown. Moreover, each possible event occurrence is unique regarding its time of occurrence. Thus, no generalization of a loop is possible regarding its temporal properties. Assuming that for each loop the maximum number of iterations is known, however, any process schema with well-nested loops can be transformed into a loop-free one by replacing each loop structure by a sequence of nested XOR blocks containing the respective number of clones of the original loop body [27, 79]. Moreover, in reality it is almost always possible to estimate the maximum number of iterations.

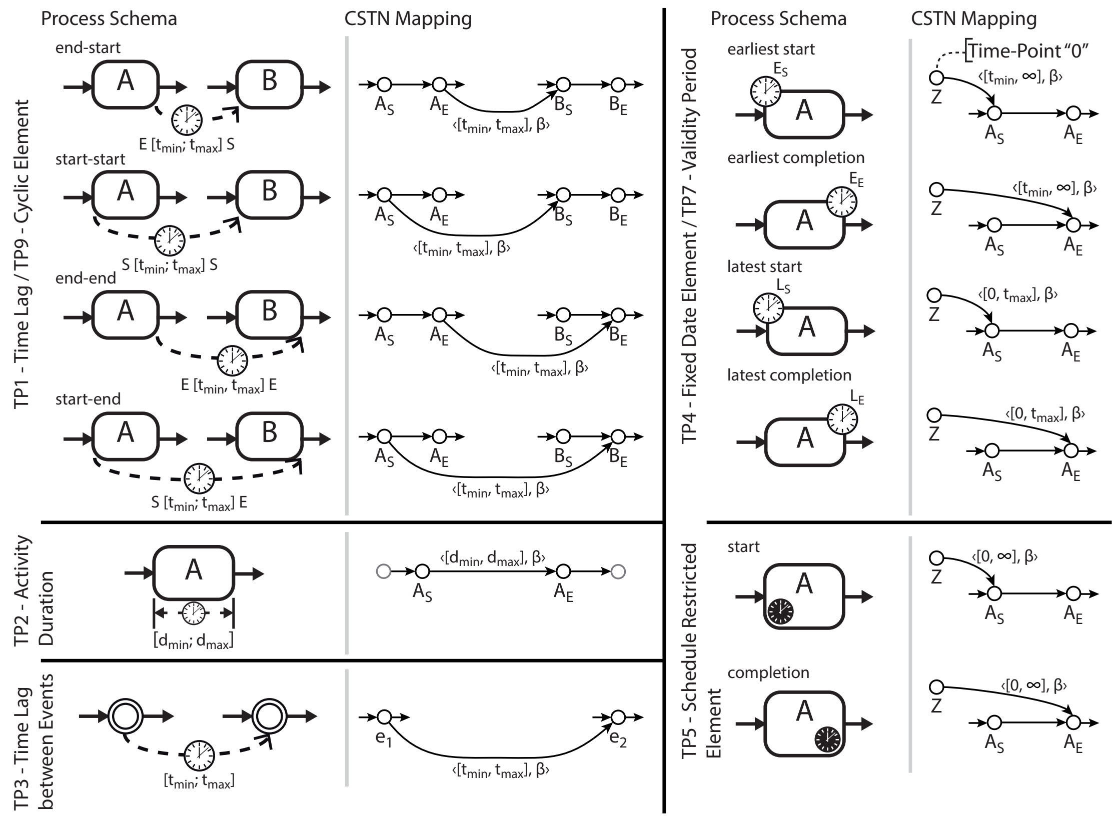


Figure 4.3: Time Patterns and their mapping to CSTN (LRW16)

Next, temporal constraints (i. e., occurrences of the time patterns) are mapped to the CSTN as depicted in Figure 4.3. Note that certain time patterns cannot be verified at design time (e. g., *Fixed Date Elements* (TP4) and *Schedule Restricted Elements* (TP5); cf. LWR14). Therefore, at design time they are mapped to an unrestricted CSTN constraint (i. e., $\langle [0, \infty]_{XY}, \beta \rangle$) in order to prepare the CSTN for execution (cf. Section 4.3). During run time the value of the CSTN constraint will be fixed as soon as the value of the respective time pattern becomes known.

A *Time-dependent Variability* (TP8) referring to the execution time of a node is equivalent to an XOR-split, whose decision rule is based on the execution time of the corresponding node, i. e., its mapping corresponds to the one of an XOR-split.

One can verify that the presented mapping indeed preserves the semantics of the respective time patterns as defined by their formal semantics (cf. LPCR13). We denote the result of this mapping as the *time model* of the time-aware process schema. As an example of this mapping consider the process schema and the corresponding time model from Figure 4.4. For a more detailed discussion of the mapping we refer to (LRW16). *Time Model*

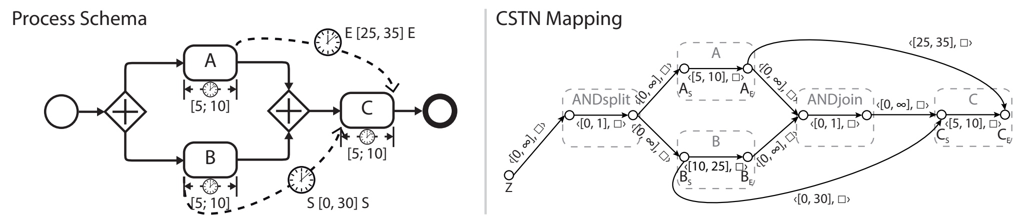


Figure 4.4: Process schema and corresponding CSTN mapping

Based on this mapping it becomes possible to verify the temporal consistency of a time-aware process. In particular, CSTN consistency is defined as follows [154]:

Definition 4.2 (Consistency of a CSTN)
Given a CSTN $N = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$, a *scenario* over set P is a function $s_P : P \to \{\mathsf{true}, \mathsf{false}\}$ that assigns a truth-value to each proposition in P.

A *solution* for CSTN N under scenario s_P corresponds to a complete set of assignments to all timepoints $X \in \mathcal{T}$ with $s_P(L(X)) = \mathsf{true}$, which satisfies all constraints $\langle [x, y]_{XY}, \beta \rangle \in \mathcal{C}$ for which $s_P(\beta) = \mathsf{true}$ holds.

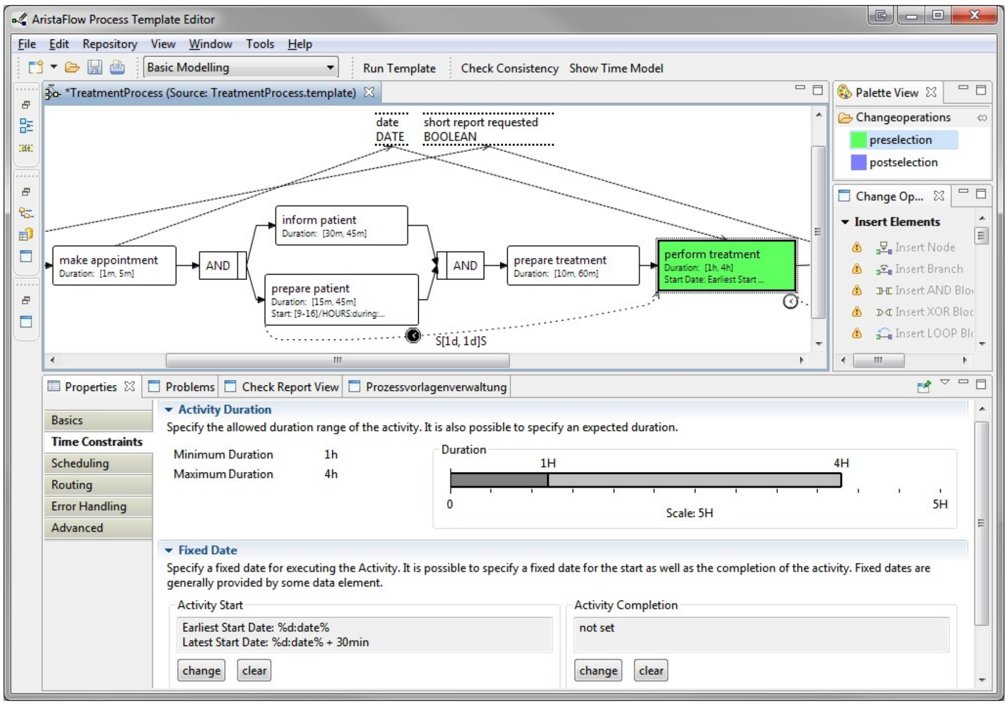
A CSTN N is called *weakly consistent* iff for each scenario s_P at least one solution exists [154], i. e., a CSTN is weakly consistent iff all of its constraints are satisfiable.

Note that a CSTN *scenario* is similar to an execution path of a process schema. Moreover, each solution of a CSTN corresponds to a temporally compliant trace of the process schema. Therefore, the CSTN corresponding to a time-aware process schema is *weakly consistent*, if for each execution path (scenario) of the process schema at least one temporally compliant temporal execution trace (solution) exists. This results in the following definition of temporal consistency of a time-aware process schema.

Definition 4.3 (Design-Time Temporal Consistency)
A time-aware process schema is denoted as *design time temporally consistent* iff the corresponding *time model* (i. e., the mapping of the process schema to CSTN) is weakly consistent.

Checking temporal consistency of a process schema can be based on existing CSTN algorithms [58, 154], which are known to be sound and complete. Note that these algorithms can derive all interdependencies between different timepoints of the CSTN; i. e., they are able to derive all interdependencies between nodes of the process schema. The resulting *minimal network* can be used for monitoring an instance of the process schema during run time (cf. Section 4.3).

Minimal Network


AristaFlow Process Template Editor
File Edit Repository View Window Tools Help
Basic Modelling
Run Template
Check Consistency
Show Time Model
*TreatmentProcess (Source: TreatmentProcess.template)
date DATE
short report requested BOOLEAN
make appointment Duration: [1m, 5m]
AND
inform patient Duration: [30m, 45m]
prepare patient Duration: [15m, 45m] Start: [9-16]/HOURS:during:...
AND
prepare treatment Duration: [10m, 60m]
perform treatment Duration: [1h, 4h] Start Date: Earliest Start ...
S[1d, 1d]S
Palette View
Changeoperations
preselection
postselection
Change Op...
Insert Elements
Insert Node
Insert Branch
Insert AND Bloc
Insert XOR Bloc
Insert LOOP Bl
Properties
Problems
Check Report View
Prozessvorlagenverwaltung
Basics
Time Constraints
Scheduling
Routing
Error Handling
Advanced
Duration
1H
4H
0
5H
Scale: 5H
change
clear
change
clear

Activity Duration

Specify the allowed duration range of the activity. It is also possible to specify an expected duration.

Minimum Duration	1h
Maximum Duration	4h

Fixed Date

Specify a fixed date for executing the Activity. It is possible to specify a fixed date for the start as well as the completion of the activity. Fixed dates are generally provided by some data element.

Activity Start

Earliest Start Date: %d:date%
Latest Start Date: %d:date% + 30min

Activity Completion

not set

Figure 4.5: The ATAPIS Process Editor

Proof-of-Concept Prototype

The ATAPIS framework has been implemented as a proof-of-concept prototype as part of the ATAPIS Toolset[2] [72]. The ATAPIS Toolset allows specifying process schemas enriched with temporal constraints (cf. Figure 4.5). In particular, it provides support for the time patterns most commonly required in practice: *Time Lags between two activities* (TP1), *Durations* (TP2) of activities and processes, *Fixed Date Elements* (TP4), *Schedule Restricted Elements* (TP5), *Validity Periods* (TP7), *Time-Dependent Variability* (TP8) based on the execution time, and *Cyclic Elements* (TP9). The resulting time-aware process schemas may then be checked for temporal consistency based on the defined pattern semantics.

4.1.3 Evaluation and Related Work

The implementation of the time patterns is based on Conditional Simple Temporal Networks (CSTN) [57]–a network based representation of Conditional Simple Temporal Problems [154]. The latter represents a special kind of Temporal Constraint Satisfaction Problem [39]—a well known problem from the artificial intelligence and planning domain.

[2]The ATAPIS Toolset, some examples and a screencast showing the toolset are available for download at dbis.info/atapis

A first sound and complete algorithm for verifying the consistency of CSTN is discussed in [154]. In turn, Hunsberger et al. [58] present the first sound and complete algorithm for checking dynamic consistency of CSTN based on constraint propagation. Besides CSTN there exists a variety of other temporal constraint problems like Simple Temporal Network with Uncertainty (STNU) [106, 108], Conditional Simple Temporal Network with Uncertainty (CSTNU) [30, 57], and Conditional Temporal Problem with Preferences (CTPP) [49]. Some of these could have been used alternatively for implementing the time patterns (cf. Section 4.2 for details on the use of CSTNU).

As discussed in Section 3.2.3, both Bettini et al. [11] and Combi et al. [27, 28] use Simple Temporal Network (STN) [39] as basic formalism for representing and reasoning about temporal constraints in processes. Bettini et al. [11] consider *Time Lags between Activities* (TP1), activity *Durations* (TP2), and *Fixed Date Elements* (TP4), whereas Combi et al. [27] also consider *Schedule Restricted Elements* (TP5). Note that STN is a simplified variant of CSTN. In particular, it holds that a CSTN $N = \langle \mathcal{T}, \mathcal{C}, \emptyset, \emptyset, \emptyset, \emptyset \rangle$ is equivalent to an STN [57]. Consequently, the translation of time-aware process schemas to STN proposed by Bettini et al. [11] and Combi et al. [27], respectively, is similar to the one chosen by us. Moreover, the loop transformation proposed by [27] is similar to the one proposed by (LRW16).

Eder et al. [43, 45, 46] use an extension of the *Critical Path Method* (CPM) [112]–called the Timed Workflow Graph (cf. Section 3.2.3). [46] considers *Time Lags between Activities* (TP1), activity *Durations* (TP2), *Fixed Date Elements* (TP4), and *Schedule Restricted Elements* (TP5). Essentially, a Timed Workflow Graph is the same as the process schema. Each activity has a fixed duration and is augmented by two values for its earliest and latest completion time. To handle XOR-splits and -joins, the Timed Workflow Graph is extended in [43] by splitting up the earliest and latest completion time into the earliest/latest completion of the best/worst case. These four values are calculated for each activity by using a modified version of the CPM algorithm. In turn, [46] proposes to unfold a Timed Workflow Graph at each XOR-join by duplicating the remaining Timed Workflow Graph for each possible path.

Marjanovic et al. [101] define a conceptual model for temporal constraints on a process schema. When taking the time patterns as benchmark, [101] considers *Time Lags between Activities* (TP1), activity and process *Durations* (TP2), and *Fixed Date Elements* (TP4). Further, a set of rules for verifying time-aware process schemas is presented.

Cicirelli et al. [22] present time stream Petri net (TSPN) [41] representations for selected time pattern variants. For analyzing the resulting network, [22] proposes the use of either a suitable transformation to timed automata [3] and model checking tools like UPPAAL [87], or—if respective models become too large—actor-based simulation techniques based on Parallel Discrete Events Systems [21].

Moreover, Maria et al. [37] and Maggi et al. [96] use timed automata [3], in combination with model checking techniques like Metric Temporal Logic (MTL) [68], for verifying the satisfiability of a time-aware process schema. Similarly, other model checking techniques like TPTL [4] and ECL [52] may be used (see [51] for a survey on linear-time timed temporal logics). Finally, David et al. [35] present a framework for the formal verification of real-time UML statecharts based on timed automata and model checking techniques.

Most of these techniques may be used for verifying the temporal consistency of time-aware processes with different pros and cons. However, existing approaches (e. g., based on temporal constraint satisfaction problems [11, 27] and timed automata [37, 96]) only partially cover the time patterns and their variants. We have chosen CSTN for implementing the time patterns as it is very similar to the temporal execution traces used for defining the formal pattern semantics and at the same time has lower computational complexity compared to most other techniques. Moreover, CSTN offer promising perspectives for supporting the monitoring of time-aware process instances as well as some advanced time support features (cf. Section 4.3).

4.1.4 Discussion and Outlook

The use case analysis we performed as part of (LWR14) and the definition of the time pattern semantics in (LRW16) revealed the need for a comprehensive design-time support of time-aware processes. In particular, to enable a robust and error-free process execution, soundness of respective process schemas must be ensured. For time-aware processes this encompasses ensuring the temporal consistency of the process schema. To this end, the ATAPIS framework presented by (LRW16) provides a comprehensive and generic framework and corresponding techniques for verifying the temporal consistency of time-aware process schemas based on the formal semantics of the time patterns. In particular, we present a suitable transformation of time-aware process schemas to CSTN that preserves the semantics of the time patterns. Based on this transformation provably sound and complete checking algorithms for CSTN can be used to verify the temporal consistency of time-aware process schemas. Moreover, respective CSTN algorithms enable us to derive implicit temporal constraints between any pair of timepoints of the CSTN.

During run time the CSTN transformation provided by the ATAPIS framework may be used for scheduling and monitoring corresponding process instances (cf. Section 4.3). Moreover, the derived implicit temporal constraints may be used by the PAIS to ensure observance of any explicit temporal constraints as well as to predict possible future violations of the latter.

Current limitations of the ATAPIS framework concern the use of weak consistency of CSTN for defining temporal consistency of time-aware processes and the relaxed treatment of activity durations. In particular, a weakly consistent CSTN might require a-priori knowledge about the execution paths to be taken in the future to be able to guarantee the successful completion of a process instance. However, such knowledge might not be available in practice. Yet, stronger consistency notions of CSTN (e. g., dynamic consistency [154]) are incompatible with certain time patterns (e. g., *TP8: Time-dependent Variability*). Regarding activities, in turn, the ATAPIS framework currently assumes that their actual duration may be restricted by the PAIS. However, in reality this might not be the case as the duration of activities is uncertain and mostly depends on the performer of the activity, the actual task to be accomplished, and the environment. Both challenges are picked up in Section 4.2.

4.2 Controllability of Time-Aware Processes

The content of this section was published as follows:

LPCR13 A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controllability of time-aware processes at run time. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences — Proceedings of the 21st International Conference on Cooperative Information Systems (CoopIS'13)*, number 8185 in Lecture Notes in Computer Science, pages 39–56. Springer, September 2013. DOI: 10.1007/978-3-642-41030-7_4

The original article is added in Appendix A.3.

4.2.1 Problem Description

Contingent Duration

Regarding the run-time support of time-aware processes it is noteworthy that the duration of an activity executed in the context of a process instance is usually *contingent*. Indeed, although it is possible to specify a duration range for an activity, its actual duration is subjected to the environment and only becomes known after its completion; i. e., it cannot be controlled by the PAIS. As an example consider a surgery: Although it is possible to specify a duration range for such activity (e. g., one to two hours), the effective duration only becomes known after the surgery has finished as it depends on the complexity of the procedure as well as any possible minor or major complications that might emerge during the procedure. This needs to be taken into account when verifying the *temporal consistency*[3] of at time-aware process schema at design time. In particular, it should be possible to successfully complete an instance of a process schema for all allowed durations of its activities, satisfying all temporal constraints. The latter implies that it must be possible to execute a process instance without ever having to restrict the duration of an activity to satisfy one of the other temporal constraints. Yet, at the same time activity durations are usually specified reasonably generous and may thus still be restricted to some limited extend. Note that, although this is more strict than the semantics of pattern *Duration* (TP2) as defined in (LRW16), it does not conflict with respective definitions as these need to be applicable in all scenarios and, hence, represent the lowest common denominator of all possible cases.

As additional challenge, temporal consistency needs to be ensured for all possible execution paths of a process schema (e. g., due to exclusive choices or loops). In particular, each decision taken at run time may lead to different temporal properties of the remaining process. This is particularly challenging, as the temporal constraints of one execution path may affect parts of the process executed prior to the decision which execution path shall actually be taken (cf. Example 4.2).

[3]The referenced paper uses the term "controllability" instead of temporal consistency as will be explained further on.

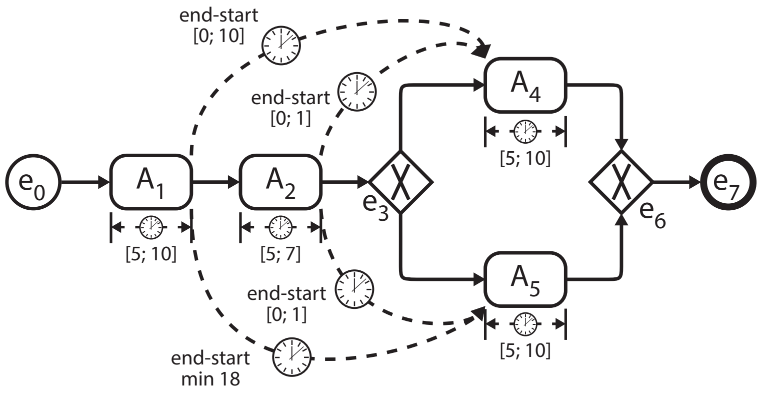


Figure 4.6: Conflicting alternative execution paths

Example 4.2 (Conflicting alternative execution paths)
Figure 4.6 depicts a process schema consisting of four activities and two alternative execution paths. Note that this process schema is temporally consistent according to Definition 4.3. However, its successful execution requires a-priori knowledge about the execution path to be taken at XOR-split e_3.

First, consider the upper execution path (i. e., the one containing A_4). In this case, one may conclude that, to satisfy the maximum time lag of 10 between activities A_1 and A_4, activity A_2 may be started no later than 5 after the completion of A_1. In particular, if A_2 is started 5 after A_1 and take its minimum duration (i. e., 5), A_4 may be started the earliest 10 after A_1, which still satisfies the maximum time lag of 10 between A_1 and A_4.

In turn, considering the lower path, one may conclude, that to satisfy the minimum time lag of 18 between activities A_1 and A_5, activity A_2 may be started the earliest 10 after A_1. In detail, if A_2 is started 10 after completing A_1 and takes its maximum duration of 7, then—considering the maximum time lag of 1 between A_2 and A_5—A_5 must be started the latest 18 after the completion of A_1, which still satisfies the minimum time lag of 18 between A_1 and A_5.

Hence, when starting activity A_2 one needs to know which execution path will be taken in order to ensure that all remaining temporal constraints can be satisfied. Usually, however, such information is not available prior to the execution of e_3.

Thus, at design time it must be ensured than any execution decision made is compatible with all possible future execution paths, i. e., the decision when to execute a particular activity cannot rely on any a-priori knowledge about future execution paths.

4.2.2 Scientific Contribution

(LPCR13) builds upon the ATAPIS framework discussed in Section 4.1. (LPCR13) extends respective results in two ways: *First,* a basic model for specifying time-aware process schemas exhibiting the properties outline above is presented, i. e., the model specifically considers the contingent, but restrictable nature of activity durations. *Second,* the referenced paper presents a mapping of time-aware processes to *Conditional Simple Temporal Network with Uncertainty* (CSTNU) [30, 57], which allows checking the *dynamic controllability* of respective process schemas at design time. In the context of CSTNUs, controllability refers to the ability of executing a network for all allowed durations of its contingent constraints (i. e., activities) and satisfying all temporal constraints.

Dynamic Controllability

Time-Aware Process Schemas

To set a focus, the referenced paper specifically considers the time patterns most relevant in practice according to our analysis (cf. Section 3.1.2): *Time Lags between Activities* (TP1), *Durations* (TP2), *Fixed Date Elements* (TP4), and *Cyclic Elements* (TP9).

In the context of *activity durations*, a closer analysis of real-world cases reveals another important property regarding the contingent nature of activity durations. In particular, many real-world processes turn out to be not temporally consistent if activity durations are considered to be completely contingent. One of the main reasons for this is that, although in theory activity durations must not be restricted by the PAIS as respective durations are decided by the environment, in practice the durations specified for activities usually represent worst case estimates; i. e., they cover cases with an exceptionally long duration which only very rarely occur in practice. Moreover, in practice these cases are usually dealt with on a case-to-case basis. Finally, if necessary, execution times of most activities may be restricted to some extend.

In order to represent *activity durations* more universally, therefore, (LPCR13) proposes to represent them in terms of *restrictable time intervals* $[[MinD_C, MaxD_C]\ MaxD_F]]$ $(1 \leq MinD_C \leq MaxD_C \leq MaxD_F)$. In this context, $MaxD_F$ represents the flexible maximum duration of the activity (i. e., the worst case). If necessary, this may be restricted up to the contingent minimum and maximum duration range $[MinD_C, MaxD_C]$, which, in any case, must be available for executing the activity.

Restrictable Time Interval

The remaining temporal constraints considered by this work are defined similarly to what has been discussed in Section 4.1.2. We do not repeat respective descriptions here.

Controllability of Time-Aware Process Schemas

In order to verify the temporal consistency of a time-aware process schema, considering the contingent nature of its activities as well as the impact alternative execution paths

CSTNU

may have, (LPCR13) proposes the use of *Conditional Simple Temporal Network with Uncertainty* (CSTNU) [30, 57]. This choice has been made for three reasons: (1) it is preferable to exploit a well founded model of extended temporal constraint representation (including corresponding algorithms) instead of developing native algorithms solving the same basic problem, (2) CSTNU is the only model being able to represent and manage conditional execution paths and contingent durations at the same time, and (3) CSTNU represents an extension of CSTN used for formally defining time pattern semantics. Hence, everything that applies for CSTN (cf. Section 4.1.2) applies for CSTNU as well.

Formally, a CSTNU can be defined as follows (cf. LPCR13):

Definition 4.4 (Conditional Simple Temporal Network with Uncertainty)
A Conditional Simple Temporal Network with Uncertainty (CSTNU) [57] 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 (cf. Definition 4.1).
- $\mathcal{L}$ is a set of *contingent links*; each contingent link $c_{AC} \in \mathcal{L}$ has the form $c_{AC} = (A, x, y, C)$, where $A, C \in \mathcal{T}$ and $0 < x < y < \infty$.
- for each $(A, x, y, C) \in \mathcal{L}$ it holds $L(A) = L(C)$ and $\langle [x, y]_{AC}, L(A) \rangle \in \mathcal{C}$.

Contingent Link

A *contingent link* (A, x, y, C) represents an uncontrollable, but bounded temporal interval. C is called the *contingent timepoint*, and A its *activation timepoint*. A contingent link can be interpreted as follows: Once A is executed, C is guaranteed to execute such that $C - A \in [x, y]$ holds. However, the particular time at which C occurs cannot be controlled, but only be observed when it happens.

Dynamic Controllability

In [30], Combi et al. present a sound algorithm that allows determining the *dynamic controllability* of a CSTNU.

Definition 4.5 (Dynamic Controllability of CSTNU)
A CSTNU is called *dynamically controllable* [57] if it is possible to execute it for any allowed durations of its contingent links and any possible execution scenario (cf. Definition 4.2) without violating any of the temporal constraints or requiring any a-priori knowledge about the execution.

Based on CSTNU, (LPCR13) shows that a time-aware process schema can be mapped to a corresponding CSTNU such that all temporal features of the process schema are represented in the CSTNU. In particular, we provide an appropriate mapping of each process construct to an equivalent CSTNU fragment. Most interestingly, each activity A with restrictable duration $[[x_C, y_C]\ y_F]]$ corresponds to three timepoints A_S, A_C, and A_E in the CSTNU. A_S represents the start timepoint (i. e., start event) of the activity and A_E its end timepoint (i. e., end event). In turn, A_C is an internal timepoint that is only used for checking controllability of the CSTNU, but is not considered during

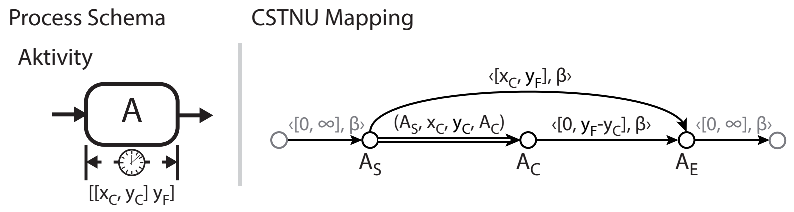


Figure 4.7: An activity with a restrictable duration and its CSTNU translation.

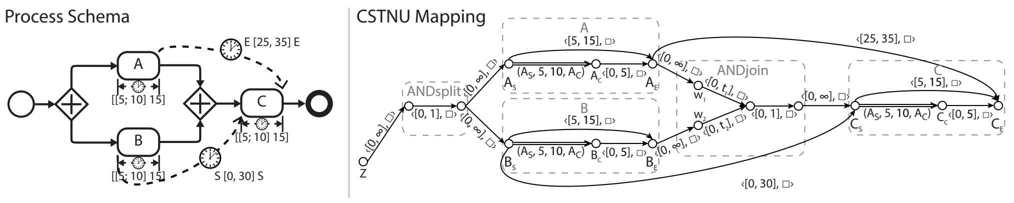


Figure 4.8: Process schema and corresponding CSTNU mapping

execution. Respective timepoints are linked by appropriate constraints representing the given duration of the activity as shown in Figure 4.7.[4] In particular, timepoint A_C represents the uncontrollable ending point of *A* and is bounded by contingent link (A_S, x_C, y_C, A_C) with respect to start timepoint A_S. In turn, A_E is the controllable end timepoint, bound to A_S by the simple temporal constraint $\langle [x_C, y_F]_{A_S A_E}, \beta \rangle$, which allows the checking algorithm to consider the flexible maximum duration y_F. Finally, the lower bound of the constraint between A_C and A_E is set to 0 in order to indicate that A_E has to be executed after A_C while the upper bound is set to $y_F - y_C$ in order to allow for the flexible maximum duration.[5] This mapping fully represents the restrictable duration range of an activity as previously described.

In turn, the mapping of other process constructs is similar to the one discussed in Section 4.1.2, and hence not considered here. Note that there are minor differences in the mapping of the AND-join connector which that been introduced to enforce an earliest execution strategy of AND-join connectors as applied by most PAIS (cf. Figure 4.8). However, these differences have no influence on the dynamic controllability of the CSTNU and, hence, the temporal consistency of the process schema. As an example for this mapping consider the process schema and the corresponding time model depicted in Figure 4.8.

Based on the respective CSTNU mapping, we can extend the notion of design-time temporal consistency (cf. Definition 4.3) to consider contingent activity durations and alternative execution paths.

[4]For the sake of consistency, this thesis uses a network-based representation for CSTNU, whilst (LPCR13) uses a distance-graph-based representation. Note that both representations are fully equivalent.

[5]Note that the original publication contains a write error, where $y_F - x_C$ instead of $y_F - y_C$ is used for the upper bound of the temporal constraint between A_C and A_E. When using this erroneous formula, it allows for more flexibility with respect to the minimum contingent duration range than supposed.

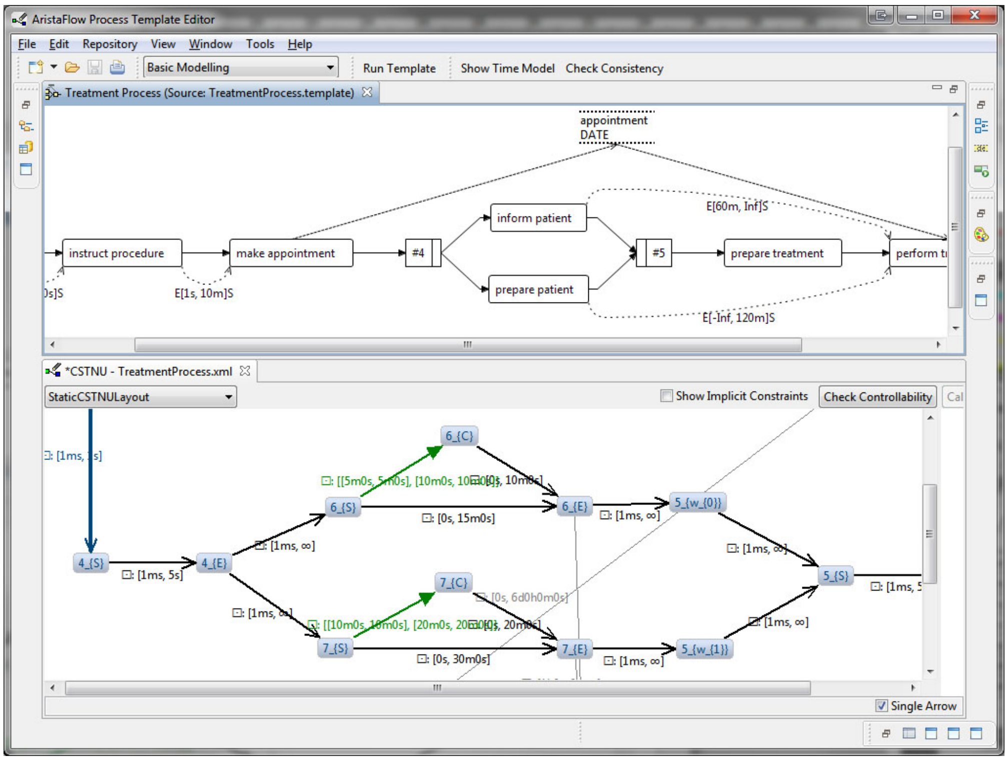


Figure 4.9: Checking temporal consistency of a process schema using the ATAPIS process editor

Definition 4.6 (Design-Time Temporal Consistency with Contingent Durations)
A process schema S with temporal constraints and contingent activity durations is denoted as *design-time temporally consistent* iff the corresponding CSTNU is dynamically controllable.

Proof-of-Concept Prototype

The concepts presented in (LPCR13) have been implemented as a proof-of-concept prototype as part of the ATAPIS Toolset [72]. This prototype allows creating time-aware process schemas based on the basic elements defined by (LPCR13). Respective, process schemas can then be automatically transformed to a corresponding CSTNU and be checked for dynamic controllability (i. e., temporal consistency). Altogether the prototype demonstrates the practical feasibility of the proposed transformation and respective algorithms. Figure 4.9 depicts a screenshot of the ATAPIS process editor showing a time-aware process schema at the top and its corresponding CSTNU at the bottom.

4.2.3 Evaluation and Related Work

Bettini et al. [11] consider the contingent nature of process activities as well. In particular, they introduce the concept of a *free schedule* for a time-aware process schema: A schedule is free if it is possible to statically fix the start time of all activities of a process schema before starting the execution of corresponding process instances, without constraining the durations of respective activities, and while satisfying all constraints [11]. The concept of free schedule is close to the one of controllability. In particular, a free schedule corresponds to a controllable path of the process schema. Note that the opposite is not necessarily true, as the start times of the activities of a dynamically controllable path may be dynamically decided during run time.

The concepts of contingent activity durations and controllability have been mainly investigated in the artificial intelligence and planning area in connection with temporal constraint networks. The concept of contingent durations was first discussed by Vidal et al. [163, 164] and the *Simple Temporal Problem under Uncertainty* (STPU) in combination with a basic notion of controllability is presented. In turn, Morris et al. [106, 108] propose an extension of *Simple Temporal Network* (STN) [39], the *Simple Temporal Network with Uncertainty* (STNU) to solve the STPU. Subsequently they propose multiple sound and complete algorithms for checking the dynamic controllability of STNU in pseudo-polynomial time [105, 107, 108]. Combi et al. [26] were the first group transferring the concept of controllability to time-aware processes. In particular, [26] provides inference rules to verify the controllability of sequential and parallel paths of a process schema—alternative paths are not considered by this work.

Recently, Hunsberger et al. [57] merged CSTN and STNU to *Conditional Simple Temporal Network with Uncertainty* (CSTNU), supporting both alternative execution paths and contingent duration constraints. Moreover, they extended the notion of dynamic controllability in a suitable way. In turn, Combi et al. [30, 31] presented the first sound, but not complete algorithm for checking dynamic controllability of a CSTNU. Moreover, Cimatti et al. [23] presented the first sound and complete algorithm for verifying dynamic controllability of a CSTNU by showing that CSTNU can be translated to Timed Game Automata [97] and then be verified using model checking tools like UPPAAL-Tiga [9].

(LPCR13) is the first work to apply CSTNU in the context of time-aware processes to check temporal consistency of the latter. Moreover, it is the only work considering the contingent, but restrictable nature of process activities and proposing a suitable technique for checking temporal consistency of corresponding time-aware process schemas.

4.2.4 Discussion and Outlook

(LPCR13) considers fundamental requirements for the comprehensive design-time support of time-aware processes. *First,* we define a set of basic elements for modeling time-aware process schemas, which specifically consider the contingent, but restrictable nature of activity durations and allows for a flexible execution of related process instances. *Second,* a transformation of time-aware process schemas to CSTNU is presented for checking the temporal consistency of time-aware process schemas with contingent activity durations and alternative execution paths at design time.

To the best of our knowledge, (LPCR13) has been the first work to comprehensively consider contingent activity durations and alternative execution paths in an integrated way. As limitation of the presented work, to set a focus, only a limited set of time patterns is considered. However, the results presented in (LRW16) (cf. Section 4.1.2) show that the approach presented by the referenced paper can be easily extended to consider a wider range of time patterns as well. Yet, the support of certain time pattern variants, which might conflict with the notion of dynamic controllability (e. g., *TP8: Time-dependent Variability*), still require further investigation. Moreover, a closer analysis of selected use cases has revealed that in some scenarios (e. g., in the context of modularized processes [84, 85]) the proposed restrictable activity duration range and its CSTNU transformation may still be too limited.

4.3 Executing Time-Aware Processes

The content of this section was published in the following articles:

LPCR13	A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controllability of time-aware processes at run time. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences — Proceedings of the 21st International Conference on Cooperative Information Systems (CoopIS'13)*, number 8185 in Lecture Notes in Computer Science, pages 39–56. Springer, September 2013. DOI: 10.1007/978-3-642-41030-7_4
LRW16	A. Lanz, M. Reichert, and B. Weber. Process time patterns: A formal foundation. *Information Systems*, 57:28–68, 2016. DOI: 10.1016/j.is.2015.10.002

The original articles are added in Appendices A.2 and A.3.

4.3.1 Problem Description

Ensuring temporal consistency of a time-aware process schema solely at design time is not sufficient. In particular, during the execution of a process instance, temporal constraints need to be continuously updated according to the actual durations of already executed activities as well as the control-flow decisions made during run time. Moreover, not all temporal constraints may be fully known at design time. For example, the value of a *Fixed Date Element* (TP4; e. g., an appointment) only becomes known during run time and is specific to each process instance. Finally, temporal constraints that cannot be fully checked at design time (e. g., *TP5: Schedule Restricted Elements*) can now be verified based on the actual execution time frames of the process instance and corresponding activity instances.

Hence, for time-aware processes, it becomes necessary to continuously monitor and update their temporal constraints during run time and to re-verify temporal consistency of respective process instances. Particularly, note that any process instance is unique with respect to its temporal properties (e. g., execution time frames, actual activity durations etc.).

4.3.2 Scientific Contribution

Temporal consistency of a process schema must be checked both at design and run time. At design time, such a consistency check allows guaranteeing that the design-time phase is sound as any process instance may be executed meeting the given temporal constraints. At run time, the consistency check updates the time model according to the real durations of already executed activities, to the possible fixed date constraints, and to the current execution path. In particular, temporal consistency has to be verified after the completion of each activity.

Instance Time Model

To be able to monitor the temporal consistency of a process instance, when creating it, the time model (i. e., CSTNU transformation) derived at design time is cloned. This *instance time model* is then updated according to the starting time of the process instance. Next, all temporal constraints which become known at process creation time are considered by updating respective constraint(s) in the network. Regarding a *Fixed Date Element*, for example, the CSTNU constraint inserted at design time is updated to the actual value of the pattern occurrence. Moreover, *Schedule Restricted Elements* are considered by restricting the corresponding CSTNU constraint such that its bounds lie within the first and last possible time slot of the respective schedule. Finally, a consistency check is performed to update any implicit constraints. This determines the time frame for starting the first activity, which may then be used by the process execution engine.

Note that the ATAPIS framework employs a preemptive strategy for executing and monitoring activities. In particular, it monitors the start and execution of all activities to detect and, if possible, prevent constraint violations. In this context, the implicit temporal constraints, as derived by the CSTNU checking algorithm, are used to discover future violations of a temporal constraint not directly related to the current activity (e. g., future deadlines that can no longer be met).

When completing an activity, the CSTNU instance must be updated to the actual duration of the activity. Moreover, the activity may provide the parameter values for some other temporal constraints (e. g., the date value of a fixed date element). In turn, each time an XOR-split is executed, the CSTNU instance must be updated by removing all nodes and edges belonging to skipped XOR branches. In particular, when executing an XOR-split the corresponding observation timepoint is also executed, determining the truth value of the associated propositional letter (cf. Definition 4.1). Therefore, the execution scenario (cf. Definition 4.2) is updated and all nodes/edges not consistent with it are removed. Each time the CSTNU instance is updated, a controllability check must be performed to propagate any changes.

Algorithm 1 shows the pseudo code of the algorithm *TimeAwareProcessControllabilityCheck* as proposed by (LPCR13). It updates the CSTNU instance as described and subsequently checks the dynamic controllability of the network. Note that the only possible reasons for the network becoming uncontrollable in Line 11 is that either the execution of an activity takes longer than permitted or the date value set for a fixed date element is not consistent with the CSTNU. In both cases, a time-specific exception handling (e. g., escalations) should be triggered as the ultimate cause for such error is out of control of the PAIS.

Algorithm 1: TimeAwareProcessControllabilityCheck(S, N, e)

input : Process instance S with time model N and the event e which has just occurred
output: Whether or S is temporally consistent

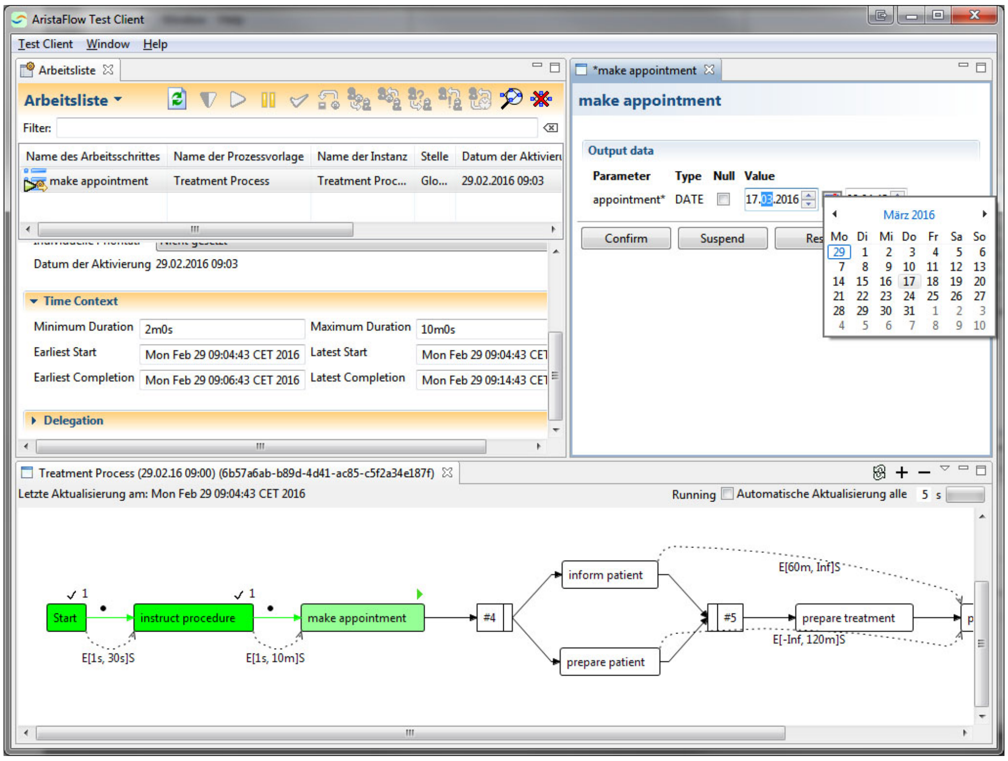
```
if ($e$ == "end of activity" $\mathsf{A_i}$) then
    $d_i$ = real duration of $\mathsf{A_i}$;
    Update all constraints in $N$ involving $\mathsf{A_i}$ using $d_i$;
    foreach $c_{ij}$ = constraint value known after the execution of $\mathsf{A_i}$ do
        if ($c_{ij} \neq null$) then
            Update all constraints in $N$ requiring $c_{ij}$;
        end
    end
    Execute CSTNU consistency check on $N$;
    if ($S$ is not temporally consistent) then
        Throw an exception;
    end
end
if ($e$ == "end of XOR-split" $\mathsf{X_i}$) then
    $d_i$ = real duration of $\mathsf{X_i}$;
    Update all constraints in $N$ involving $\mathsf{X_i}$ using $d_i$;
    $b_i$ = selected branch;
    Remove all branches (edges and nodes) $\neq b_i$;
    Execute CSTNU consistency check on $N$;
end
return $S$ temporally consistent;
```

Proof-of-Concept Prototype

The concepts presented in (LPCR13; LRW16) haven been implemented as a proof-of-concept prototype as part of the ATAPIS Toolset [72]. This prototype enables us to simulate the execution of time-aware process instances and to monitor their temporal consistency based on the presented algorithm. Figure 4.10 depicts a screenshot of the ATAPIS client showing a time-aware process schema currently being executed at the bottom and the currently active activity and its temporal constraints on the left.

4.3.3 Evaluation and Related Work

Most approaches considering temporal constraints for business processes focus on design-time issues like the modeling and verification of time-aware process schemas [11, 27, 45, 101, 175] (cf. Section 4.1.3 for a more detailed discussion of respective approaches). To the best of our knowledge, the approach presented by Eder et al. [44] is the only one explicitly considering run-time aspects of time-aware processes. In particular, [44] suggests a



Name des Arbeitsschrittes	Name der Prozessvorlage	Name der Instanz	Stelle	Datum der Aktivieru
make appointment	Treatment Process	Treatment Proc...	Glo...	29.02.2016 09:03

Datum der Aktivierung 29.02.2016 09:03

Time Context

Minimum Duration	2m0s	Maximum Duration	10m0s
Earliest Start	Mon Feb 29 09:04:43 CET 2016	Latest Start	Mon Feb 29 09:04:43 CET
Earliest Completion	Mon Feb 29 09:06:43 CET 2016	Latest Completion	Mon Feb 29 09:14:43 CET

Parameter	Type	Null	Value
appointment*	DATE	☐	17.03.2016

März 2016

Mo	Di	Mi	Do	Fr	Sa	So
29	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

Treatment Process (29.02.16 09:00) (6b57a6ab-b89d-4d41-ac85-c5f2a34e187f)

Letzte Aktualisierung am: Mon Feb 29 09:04:43 CET 2016

Automatische Aktualisierung alle 5 s

Figure 4.10: Simulating the execution of a time-aware process using the ATAPIS client

basic run-time support by calculating “internal deadlines” for each activity based on the available temporal information. In this context, activity durations correspond to either the minimum, maximum or most frequent duration of the respective activity (i. e., they are represented by a single, fixed value). Moreover, the calculation of the internal deadlines assumes that the value of any *Fixed Date Element* is known when creating the process instance; i. e., unlike in (LPCR13; LRW16), setting the particular date during run time is not considered. Based on the calculated deadlines, each process instance is associated with one out of three different run-time states—green, yellow, and red—depending on how big the current threat for missing a deadline looks like for the respective process instance. The assessment of this threat is done solely based on the pre-calculated deadlines and some pre-defined threshold values, i. e., re-calculation of the deadlines during run time is not carried out.

In [55, 56], Hunsberger proposes two algorithms for the efficient execution of dynamically controllable STNU. Both guarantee that the execution of a dynamically controllable STNU will complete without violating any of the temporal constraints. However, respective algorithms cannot directly be applied to the execution of time-aware processes as they assume that each non-contingent timepoint is executed by the system at the earliest possible time (i. e., as soon as all temporal constraints involving the timepoint can be satisfied). For time-aware processes this cannot be enforced, as the execution time of activities is generally not decided by the PAIS itself, but by its users. Nevertheless, respective algorithms provide interesting insights for the further development and optimization of the execution algorithm proposed by (LPCR13).

Kumar et al. [69] introduce the concept of controlled constraint violations as a way to successfully complete a process instance that has run into troubles by minimizing the total penalty resulting from these violations. In this context, they propose an approach for checking temporal consistency of a time-aware process schema by mapping it to a general Constraint Satisfaction Problem (CSP) [38] and solving the latter for each possible execution path using the CSP-solver CPLEX [60]. To allow for controlled violations of the temporal constraints, the CSP is extended by a relaxation variable for each constraint determining the amount by which the constraint shall be relaxed. This approach is similar—although in the opposite sense—to the restrictable time intervals proposed by (LPCR13).

4.3.4 Discussion and Outlook

To solely verify time-aware process schemas at design time is neither sufficient nor completely possible. In particular, (LWR14) has shown that certain time patterns cannot be verified at design time, as they are specific to each process instance. Moreover, run-time specific information like real activity durations and execution decisions must be considered to ensure that no constraint is violated during run time.

Therefore, (LPCR13; LRW16) consider fundamental requirements for the run-time support of time-aware processes. In particular, we demonstrate how CSTNU can be used for ensuring the temporal consistency of time-aware process instances during run time. Moreover, an abstract algorithm for checking and monitoring temporal consistency during run time is presented and its complexity is discussed (cf. LPCR13).

To the best of our knowledge (LPCR13; LRW16) have been the first works to comprehensively consider contingent activity durations, alternative execution paths and run-time support of time-aware process instances. In the future, we expect that the approach presented by the referenced papers will be used to further investigate run-time support of time-aware processes. In particular, many open issues remain regarding the proper support of time-aware process instances in PAIS (e. g., how to enforce temporal constraints and what shall happen if a temporal constraint can no longer be satisfied).

5 Dealing with Changes of Time-Aware Processes

Lost time is never found again.

(Benjamin Franklin, 1706–1790)

The content of this section was published as follows:

LR14 A. Lanz and M. Reichert. Dealing with changes of time-aware processes. In *Business Process Management — Proceedings of the 12th International Conference on Business Process Management (BPM'14)*, volume 8659 of *Lecture Notes in Computer Science*, pages 217–233. Springer, September 2014. DOI: `10.1007/978-3-319-10172-9_14`

The original article is contained in Appendix A.4.

A more detailed discussion of the presented change operations as well as additional change operations, and the proof of the main theorem of the paper, have been published in a technical report [70].

5.1 Problem Description

As process execution does not always stick to the plan, processes need to be flexible to cope with unforeseen events during run time [123]. To meet these challenges *adaptive PAIS* have been developed which allow a process instance to be dynamically modified

Change Operation

during run time using a set of *change operations* [123, 126, 130]. Respective change operations abstract from low-level *change primitives* (e. g., inserting an edge or node) and ensure that, when being applied to a sound process schema, the modified process schema will be structurally and behaviorally sound as well [123, 131] (cf. Section 2.3).

Time-aware processes, in turn, need to be even more flexible, as time can neither be slowed down nor stopped. Therefore, in practice it is common that deadlines have to be re-scheduled or temporal constraints be dynamically adapted to be able to successfully complete a process instance that has run into difficulties. Moreover, in certain scenarios it may become necessary to structurally change a time-aware process instance (e. g., by moving, deleting or inserting an activity) to cope with unforeseen events or delays during run time.

While both structural and behavioral soundness in the context of change operations have been extensively studied in literature [48, 128, 131–134, 156] temporal constraints and temporal consistency have not been considered in this context so far. When dynamically modifying a time-aware process instance it must be ensured that the resulting process instance remains temporally consistent; i. e., it may still be completed without violating any of its temporal constraints. Hence, for each *change operation* suitable pre- and post-conditions must be provided that enable us to evaluate the applicability of a change operation and to guarantee for the temporal consistency of the changed process instance. Moreover, it is important that respective change operations and algorithms work as efficiently as possible in order to not cause any significant delays that may threaten the successful completion of the process instances. The latter even becomes crucial in the context of *process evolution*, where a possibly large set of process instances needs to be migrated on-the-fly to a changed process schema [123, 129, 167].

Process Evolution

5.2 Scientific Contribution

This part of the thesis extends well established process change operations with temporal constraints. In particular, (LR14) shows how temporal consistency can be efficiently ensured in the context of dynamic changes. To this end, it provides pre- and post-conditions for respective change operations that guarantee for the temporal consistency of the changed process instance. Subsequently, the referenced paper analysis the effects respective changes have on the temporal constraints of a process instance. Based on this analysis, it then provides a means to significantly reduce the complexity when applying multiple change operations within the same *change transaction* (e. g., in the context of process evolution).

Change Transaction

Preliminaries

As discussed in Section 4.1.2, a time-aware process schema[1] is *temporally consistent* if the corresponding CSTN is weakly consistent. Moreover, for each weakly consistent CSTN there exists an equivalent representation being minimal with respect to its temporal constraints, called the *minimal network* (cf. Definition 4 in LR14). More formally:

Minimal Network

Definition 5.1 (Minimal Network)
The *minimal network* of a CSTN $N = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$ (cf. Definition 4.1) corresponds to the unique CSTN $M = \langle \mathcal{T}, \mathcal{C}', L, \mathcal{OT}, \mathcal{O}, P \rangle$ having the same set of solutions as N (cf. Definition 4.2), where each value allowed by any constraint $c \in \mathcal{C}'$ is part of at least one solution of N.

A minimal network for CSTN N exists iff N is weakly consistent. Moreover, the minimal network of a CSTN can be determined using most standard algorithms for checking weak consistency (e. g., [154]). The minimal network provides a restricted set of constraints: As long as the value of each timepoint is consistent with all constraints referring to it, we can guarantee that the entire CSTN is weakly consistent. Besides *explicit constraints* $c \in \mathcal{C}$, we obtain when mapping the process model to the CSTN (cf. Section 4.1.2), the minimal network contains *implicit constraints* between any pair of timepoints that may occur in the same *execution path*. These implicit constraints represent the effects the explicit constraints have on the overall CSTN (i. e., they represent interdependencies between explicit constraints). The implicit constraints are derived from the explicit ones when determining the minimal network.

As discussed in Section 4.3, when executing a process instance, the minimal network of the CSTN created at design time is cloned. This *instance time model* is then kept up-to-date with the actual temporal state of the process instance and is used to monitor and re-check temporal consistency of the instance (cf. Section 4.3).

Instance Time Model

Change Operations for Time-Aware Processes

Standard change operations adapting process instances without temporal constraints have been extensively studied in literature [123, 128, 131, 132, 134]. Based on the composition of these change operations, it becomes possible to realize more complex change patterns (e. g., move activity) as well [123, 167]. To ensure the soundness of the modified process instance and its schema, respective change operations abstract from low-level change primitives (e. g., adding an edge or node) and additionally define necessary state-specific pre- and post-conditions. Respective conditions have been extensively

[1]Note that the referenced paper uses the term process model instead of process schema. In the context of this work both terms can be used interchangeably.

Operation	Description
Control Flow	
$InsertSerial(\mathsf{N}_1, \mathsf{N}_2, \mathsf{N}_{\mathsf{new}}, [d_{min}, d_{max}])$	Inserts node $\mathsf{N}_{\mathsf{new}}$ with duration $[d_{min}, d_{max}]$ between directly succeeding nodes N_1 and N_2.
$InsertPar(\mathsf{N}_1, \mathsf{N}_2, \mathsf{N}_{\mathsf{new}}, [d_{min}, d_{max}])$	Inserts node $\mathsf{N}_{\mathsf{new}}$ with duration $[d_{min}, d_{max}]$ as well as an AND block surrounding the SESE block defined by N_1 and N_2.
$InsertCond(\mathsf{N}_1, \mathsf{N}_2, \mathsf{N}_{\mathsf{new}}, [d_{min}, d_{max}], c)$	Inserts node $\mathsf{N}_{\mathsf{new}}$ with duration $[d_{min}, d_{max}]$ and condition c as well as an XOR block between succeeding nodes N_1 and N_2.
$InsertBranch(\mathsf{G}_1, \mathsf{G}_2, c)$	Inserts an empty branch with condition c between XOR-split G_1 and XOR-join G_2.
$DeleteActivity(\mathsf{N})$	Deletes activity N.
Temporal Constraints	
$InsertTimeLag(\mathsf{N}_1, \mathsf{N}_2, type_{tl}, [t_{min}, t_{max}])$	Inserts a time lag $[t_{min}, t_{max}]$ between nodes N_1 and N_2. Thereby, $type_{tl} \in \{\text{start-start}, \text{start-end}, \text{end-start}, \text{end-end}\}$ describes whether the time lag is inserted between the start of the two activities, the start of N_1 and the end of N_2, the end of N_1 and the start of N_2, or the end of the two activities.
$InsertFDE(\mathsf{N}, type_{fde})$	Adds a fixed date element of type $type_{fde} \in \{E_S, L_S, E_E, L_E\}$ to node N.
$DeleteTimeLag(\mathsf{N}_1, \mathsf{N}_2, type_{tl})$	Deletes the time lag of type $type_{tl} \in \{\text{start-start}, \text{start-end}, \text{end-start}, \text{end-end}\}$ between nodes N_1 and N_2.
$DeleteFDE(\mathsf{N}, type_{fde})$	Deletes any fixed date element of type $type_{fde} \in \{E_S, L_S, E_E, L_E\}$ from node N.

Table 5.1: Basic change operations (LR14)

studied in literature [131], and thus will not be considered in the following. Instead, we focus on time-related aspects.

(LR14) extends a set of basic structural change operations with pre- and post-conditions required for changing time-aware process instances. Moreover, it provides additional change operations that allow modifying the temporal constraints of a time-aware process instance (e. g., inserting a *Time Lag*). Table 5.1 lists the change operations considered by the referenced paper and the accompanying technical report [70]. Altogether, these change operations allow changing a time-aware process instance, while guaranteeing the soundness of the underlying process schema as well as the temporal consistency of the changed process instance.

Note that any time-related instance-specific data (e. g., execution times of activities) is maintained by the *instance time model* corresponding to the process instance. Hence, it is easy to verify that it is sufficient to only consider the current instance time model when analyzing the temporal properties of the current and the modified process instance.

To discuss some of the issues faced when defining change operations for time-aware processes and to highlight some of the results we obtained in (LR14), first consider change

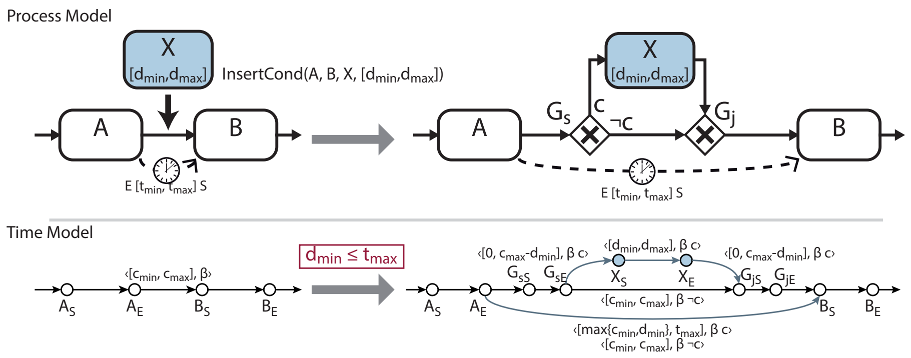

Process Model
X
[d_{min},d_{max}]
InsertCond(A, B, X, [d_{min},d_{max}])
A
B
E [t_{min}, t_{max}] S
G_s
c
¬c
G_j
Time Model
$d_{min} \leq t_{max}$
‹[c_{min}, c_{max}], β›
A_S
A_E
B_S
B_E
‹[0, c_{max}-d_{min}], β c›
‹[d_{min},d_{max}], β c›
‹[0, c_{max}-d_{min}], β c›
G_{sS}
G_{sE}
X_S
X_E
G_{jS}
G_{jE}
‹[c_{min}, c_{max}], β ¬c›
‹[max{c_{min},d_{min}}, t_{max}], β c›
‹[c_{min}, c_{max}], β ¬c›

Figure 5.1: Change operation: InsertCond

operation $InsertCond(\mathsf{N}_1, \mathsf{N}_2, \mathsf{N}_{\mathsf{new}}, [d_{min}, d_{max}], c)$. It allows inserting a node $\mathsf{N}_{\mathsf{new}}$ (e. g., an activity) with duration $[d_{min}, d_{max}]$ conditionally between directly succeeding nodes N_1 and N_2 as illustrated in Figure 5.1. This change is accomplished by first inserting XOR-split G_s and XOR-join G_j sequentially between N_1 and N_2, and then inserting $\mathsf{N}_{\mathsf{new}}$ conditionally between G_s and G_j. Moreover, the transition condition of the control edge linking G_s and $\mathsf{N}_{\mathsf{new}}$ is set to c while the one of the control edge linking G_s and G_j is set to $\neg c$. The *change primitives* required to achieve these changes are listed in Part 1 of Algorithm 2.

From a temporal point of view, it is noteworthy that, when adding XOR-split G_S and respective conditions $c/\neg c$ to the process schema, this results in a set of additional execution paths. In particular, each execution path of the original process schema, which contains N_1 and N_2, now corresponds to two execution paths, one with $c = \mathsf{false}$ representing the original execution path and one with $c = \mathsf{true}$ representing the new one (with $\mathsf{N}_{\mathsf{new}}$ being in sequence between N_1 and N_2). Hence, the temporal properties of any execution path for which $c = \mathsf{false}$ holds remain unaltered. In turn, in any execution path for which $c = \mathsf{true}$ holds, node $\mathsf{N}_{\mathsf{new}}$ is basically inserted in sequence with N_1 and N_2.

In the latter case, one can observe that the insertion of node $\mathsf{N}_{\mathsf{new}}$ might first and foremost increase the minimum time distance between N_1 and N_2 to d_{min}. By contrast, the maximum time distance between the two nodes is not affected as the added control edges do not constrain it. Accordingly, if the minimum duration d_{min} of the new node is compliant with any implicit or explicit constraint $\langle [c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta \rangle$ between the ending instant of N_1 and the starting instant of N_2 in the instance time model, the insertion of the node will not affect the temporal consistency of the process instance. In more detail, $d_{min} \leq c_{max}$ must hold to preserve the temporal consistency of the process instance after the change (cf. section `Pre` in Algorithm 2).

After updating the process schema, the mapping of the new nodes and control edges must be added to the instance time model as well (cf. 2. in Algorithm 2). In particular, this adds a new observation timepoint G_{sE} and proposition c to the time model. The labels of the temporal constraints representing $\mathsf{N}_{\mathsf{new}}$ and the two control edges connecting

Algorithm 2: InsertCond$(\mathsf{N_1}, \mathsf{N_2}, \mathsf{N_{new}}, [\mathbf{d_{min}}, \mathbf{d_{max}}], \mathbf{c})$

```
Pre   $succ(\mathsf{N_1}) = \mathsf{N_2}$,
      $\forall\langle[c_{min}, c_{max}]_{N_{1E}N_{2S}}, \beta\rangle \in \mathcal{C} : c_{max} \geq d_{min}$
Init  $\beta = L(N_{1E}) \wedge L(N_{2S})$
Post  // 1.) Update process schema:
      $RemoveEdge(\mathsf{N_1}, \mathsf{N_2})$,
      $AddNode(\mathsf{G_s}, [0,1], XOR)$, $AddNode(\mathsf{G_j}, [0,1], XOR)$,
      $AddEdge(\mathsf{N_1}, \mathsf{G_s})$, $AddEdge(\mathsf{G_s}, \mathsf{G_j})$, $AddEdge(\mathsf{G_j}, \mathsf{N_2})$,
      $AddNode(\mathsf{N_{new}}, [d_{min}, d_{max}], Activity)$, $AddEdge(\mathsf{G_s}, \mathsf{N_{new}})$, $AddEdge(\mathsf{N_{new}}, \mathsf{G_j})$,
      $UpdateCondition(\mathsf{G_s}, \mathsf{N_{new}}, c)$, $UpdateCondition(\mathsf{G_s}, \mathsf{G_j}, \neg c)$,
      // 2.) Add mapping to time model:
      $AddTimePoint(G_{sS}, \beta)$, $AddObservationTimePoint(G_{sE}, c, \beta)$,
      $AddConstraint(G_{sS}, G_{sE}, [0,1], \beta)$,
      $AddTimePoint(N_{newS}, \beta)$, $AddTimePoint(N_{newE}, \beta c)$,
      $AddConstraint(N_{newS}, N_{newE}, [d_{min}, d_{max}], \beta c)$,
      $AddTimePoint(G_{jS}, \beta)$,                    $AddTimePoint(G_{jE}, \beta)$,
      $AddConstraint(G_{jS}, G_{jE}, [0,1], \beta)$,
      $AddConstraint(N_{1E}, G_{sS}, [0,\infty], \beta)$, $AddConstraint(G_{jE}, N_{2S}, [0,\infty], \beta)$,
      $AddConstraint(N_{newE}, G_{jS}, [0,\infty], \beta c)$, $AddConstraint(G_{sE}, N_{newS}, [0,\infty], \beta c)$,
      $AddConstraint(N_{1E}, N_{newS}, [0,\infty], \beta c)$, $AddConstraint(N_{newE}, N_{2S}, [0,\infty], \beta c)$,
      $AddConstraint(G_{sE}, G_{jS}, [0,\infty], \beta\neg c)$,
      // 3.) Update minimal time model:
      $\forall\langle[c_{min}, c_{max}]_{N_{1E}N_{2S}}, \gamma\rangle \in \mathcal{C}$ :
          $AddConstraint(N_{1E}, G_{sS}, [0, c_{max} - d_{min}], \gamma)$,
          $AddConstraint(G_{sE}, N_{newS}, [0, c_{max} - d_{min}], \gamma c)$,
          $AddConstraint(N_{newE}, G_{jS}, [0, c_{max} - d_{min}], \gamma c)$,
          $AddConstraint(G_{jE}, N_{2S}, [0, c_{max} - d_{min}], \gamma)$,
          $AddConstraint(N_{1E}, N_{newS}, [0, c_{max} - d_{min}], \gamma c)$,
          $AddConstraint(N_{newE}, N_{2S}, [0, c_{max} - d_{min}], \gamma c)$,
          $AddConstraint(G_{sE}, G_{jS}, [c_{min}, c_{max}], \gamma\neg c)$,
          $UpdateConstraint(N_{1E}, N_{2S}, [c_{min}, c_{max}], \gamma\neg c)$,
          $AddConstraint(N_{1E}, N_{2S}, [\max\{c_{min}, d_{min}\}, c_{max}], \gamma c)$
```

it with $\mathsf{G_s}$ and $\mathsf{G_j}$, respectively, are set to βc with β being the conjunction of the labels of timepoints N_{1E} and N_{2S}. In turn, the label of the constraint corresponding to the control edge between $\mathsf{G_s}$ and $\mathsf{G_j}$ is set to $\beta\neg c$. Moreover, the label of any constraint $\langle[c_{min}, c_{max}]_{N_{1E}N_{2S}}, \gamma\rangle$ between the ending instant of $\mathsf{N_1}$ and the starting instant of $\mathsf{N_2}$ must be augmented by proposition $\neg c$ resulting in constraint $\langle[c_{min}, c_{max}]_{N_{1E}N_{2S}}, \gamma\neg c\rangle$. Finally, another constraint $\langle[\max\{c_{min}, d_{min}\}, c_{max}]_{N_{1E}N_{2S}}, \beta c\rangle$ containing proposition c must be added between the two timepoints. The latter corresponds to the case, where $\mathsf{N_{new}}$ is executed between the two nodes (cf. 3. in Algorithm 2). After applying this operation, the minimality of the adapted minimal time model has to be restored (e. g., by executing a consistency check). This must be accomplished before performing any other change or resuming the execution of the process instance in order to update any implicit constraints.

Algorithm 3: DeleteActivity(N)

```
Pre    $\mathsf{N}$ has no incoming or outgoing explicit temporal constraint,
Init   $\mathsf{N_p} = pred(\mathsf{N})$, $\mathsf{N_s} = succ(\mathsf{N})$,
       $c_{min}/c_{max}$ is the minimum/maximum distance of the explicit constraint between
       $\mathsf{N_p}$ and $\mathsf{N_s}$ if any, or $c_{min} = 0/c_{max} = \infty$ if there is no explicit constraint
Post   // 1.) Update process schema:
       $RemoveEdge(\mathsf{N_p}, \mathsf{N})$, $RemoveEdge(\mathsf{N}, \mathsf{N_s})$, $RemoveNode(\mathsf{N})$, $AddEdge(\mathsf{N_p}, \mathsf{N_s})$
       // 2.) Recreate minimal time model from updated process schema.
```

Algorithm 4: InsertTimeLag($\mathsf{N}_1, \mathsf{N}_2, \mathbf{type_{tl}}, [\mathbf{t_{min}}, \mathbf{t_{max}}]$)

```
Pre    $\langle I_S \rangle = \begin{cases} S & type_{tl} = \text{start-*} \\ E & type_{tl} = \text{end-*} \end{cases}, \langle I_T \rangle = \begin{cases} S & type_{tl} = \text{*-start} \\ E & type_{tl} = \text{*-end} \end{cases}$
       $(L(N1_{\langle I_S \rangle}) \wedge L(N2_{\langle I_T \rangle}))$ is satisfiable
       $\forall \langle [c_{min}, c_{max}]_{N1_{\langle I_S \rangle} N2_{\langle I_T \rangle}}, \beta \rangle \in \mathcal{C} : c_{min} \leq t_{max} \wedge t_{min} \leq c_{max}$
Post   // 1.) Update process model:
       $AddTimeLag(\mathsf{N}_1, \mathsf{N}_2, \langle I_S \rangle [t_{min}, t_{max}] \langle I_T \rangle)$
       // 2.) Add mapping to time model:
       $AddConstraint(N1_{\langle I_S \rangle}, N2_{\langle I_T \rangle}, [t_{min}, t_{max}], L(N1_E) \wedge L(N2_S))$
       // 3.) Update minimal time model:
       $\forall \langle [c_{min}, c_{max}]_{N1_E N2_S}, \beta \rangle \in \mathcal{C}$ :
           $UpdateConstraint(N1_{\langle I_S \rangle}, N2_{\langle I_T \rangle}, [\max\{c_{min}, t_{min}\}, \min\{c_{max}, t_{max}\}], \beta)$
```

For the other insert operations (i. e., *InsertSerial*, *InsertPar*, and *InsertBranch*) similar considerations apply. Hence, we will not discuss them in detail.

In turn, deleting an activity (i. e., change operation $DeleteActivity(n)$) is always possible without jeopardizing temporal consistency as temporal constraints are only removed from the time model (cf. Algorithm 3). As sole pre-condition, we require that the activity to be removed has no remaining explicit temporal constraint referring to it (e. g., time lag or fixed date element). If necessary, these have to be removed in advance using respective change operations (cf. Table 5.1).

When deleting an activity from the process instance, it is not possible to restore minimality of the modified instance time model. In particular, it is not possible to determine which of the values removed from the constraints (when establishing minimality) may now be re-added. Instead it becomes necessary to recalculate the minimal instance time model from the original one or from the process schema itself.

In terms of change operations modifying the temporal constraints of a process schema, operation $InsertTimeLag(\mathsf{N}_1, \mathsf{N}_2, type_{tl}, [t_{min}, t_{max}])$ allows adding a time lag $[t_{min}, t_{max}]$ between nodes N_1 and N_2. The instants the time lag refers to (i. e., start vs. end) are specified by parameter $type_{tl}$. In general, adding a time lag is only possible if there exists at least one execution path containing both nodes (LWR14), i. e., $L(N1_{\langle I_S \rangle}) \wedge L(N2_{\langle I_T \rangle})$ is satisfiable (cf. Algorithm 4).

The time model is then adapted by adding the mapping of the respective Time Lag (i. e., a constraint $\langle [t_{min}, t_{max}]_{N1_{\langle I_S \rangle} N2_{\langle I_T \rangle}}, L(N1_{\langle I_S \rangle}) \wedge L(N2_{\langle I_T \rangle}) \rangle \rangle$ between corresponding timepoints. This results in an update of each existing implicit constraint $\langle [c_{min}, c_{max}]_{N1_{\langle I_S \rangle} N2_{\langle I_T \rangle}}, \beta \rangle$. In turn, adding the mapping of the Time Lag is only possible if the range of the resulting constraint $[\max\{c_{min}, t_{min}\}, \min\{c_{max}, t_{max}\}]$ still permits at least one value, i. e., it allows for at least one possible solution. Accordingly, the operation may be applied if, for any implicit constraint between $N1_{\langle I_S \rangle}$ and $N2_{\langle I_T \rangle}$ it holds: $c_{min} \leq t_{max} \wedge t_{min} \leq c_{max}$. Algorithm 4 details the respective pre- and post-conditions. After updating the temporal constraints, again minimality of the adapted minimal time model must be restored.

Finally, the addition of a Fixed Date Element (i. e., operation $InsertFDE$) can be managed similarly to the insertion of a time lag. In turn, deleting a Time Lag or a Fixed Date Element (i. e., operations *DeleteTimeLag* and *DeleteFDE*) has similar pre-conditions as deleting a node (cf. operation *DeleteActivity*).

Analyzing the Effects of Change Operations

As a particular downside of the change operations discussed by (LR14), the minimality of the instance time model has to be restored after each change operation to update any implicit constraint. Only then it becomes possible to ensure that another change within the same change transaction may still be applied without violating temporal consistency of the process instance. However, calculating the minimal network of a CSTN is expensive regarding computation time. To be more precise, its complexity corresponds to $O(n^3 2^k)$ with n being the number of timepoints and k being the number of observation timepoints in the time model. Consequently, there might be significant delays when applying multiple change operations to large time-aware process schemas. This issue becomes even more pressing in the context of process schema evolution [123], when a potentially large set of process instances shall be dynamically migrated to a new process schema version.

To alleviate this issue, (LR14) proposes a method to estimate the maximum effect a particular change has on the minimal time model. Based on this, it becomes possible to decide whether or not another change operation may be applied without need to restore minimality of the instance time model first. The respective method is based on the results of Theorem 5.1 (cf. Theorem 1 in LR14), which shows how the maximum effect of any restriction of a minimal CSTN can be estimated.

Theorem 5.1
Let $M = \langle \mathcal{T}, \mathcal{C}_M, L, \mathcal{OT}, \mathcal{O}, P \rangle$ be a minimal CSTN and $M^* = \langle \mathcal{T}, \mathcal{C}_{M^*}, L, \ \mathcal{OT}, \mathcal{O}, P \rangle$ be the CSTN derived from M by replacing constraint $c_{AB} = \langle [x, y]_{AB}, \beta \rangle \in \mathcal{C}_M$ with the more restrictive constraint $c^*_{AB} = \langle [x + \sigma, y - \rho]_{AB}, \beta \rangle$; $\sigma, \rho \geq 0$; i. e., $C^*_M = C_M \setminus c_{AB} \cup \{c^*_{AB}\}$. Then: For the minimal network $N = \langle \mathcal{T}, \mathcal{C}_N, L, \mathcal{OT}, \mathcal{O}, P \rangle$ of M^* the following holds: for any constraint $c'_{XY} = \langle [x', y']_{XY}, \gamma \rangle \in \mathcal{C}_N$ the lower bound is increased by at most $\delta = \max\{\sigma, \rho\}$ and the upper bound is decreased by at most δ compared to the original constraint $c_{XY} = \langle [x, y]_{XY}, \gamma \rangle \in \mathcal{C}_M$. Formally:

$$\forall \langle [x, y]_{XY}, \gamma \rangle \in \mathcal{C}_M, \langle [x', y']_{XY}, \gamma \rangle \in \mathcal{C}_N : (x \leq x' \leq x + \delta) \wedge (y \geq y' \geq y - \delta)$$

The proof of Theorem 5.1 can be found in [70]. It is based on first showing that a CSTN can be mapped to a more simple kind of temporal problem, i. e., so called Simple Temporal Networks (STNs) [39]. Subsequently, the proof argues that the minimal network of an STN is equivalent to the all-pair-shortest-path distance matrix of the STN. Finally, the proof shows that changing any constraint in the minimal STN by δ, will reduce any shortest path in the network by at most δ as well.

To illustrate Theorem 5.1, assume that a change operation restricts a constraint $\langle [x, y]_{XY}, \beta \rangle$ in the minimal time model to $\langle [x^*, y^*]_{XY}, \beta \rangle = \langle [x + \rho, y - \sigma]_{XY}, \beta \rangle$ and afterwards minimality of the time model is restored. Theorem 5.1 now states that any constraint $\langle [u, v]_{UV}, \alpha \rangle$ in the original minimal time model is restricted to at most $\langle [u', v']_{UV}, \alpha \rangle = \langle [u+\delta, v - \delta]_{UV}, \alpha \rangle$ with $\delta = \max\{\rho, \sigma\}$ in the new minimal time model.

Reconsider change operation *InsertCond* and assume that the instance time model is adapted as described by Algorithm 2. The next step would be to restore minimality of this time model. On close consideration one can observe that the only change having an effect on the resulting minimal time model is the one restricting constraint $\langle [c_{min}, c_{max}]_{N1_E N2_S}, \beta \rangle$ between $N1_E$ and $N2_S$ to $\langle [\max\{c_{min}, d_{min}\}, c_{max}]_{N1_E N2_S}, \beta c \rangle$ (see LR14 and [70] for a more detailed discussion). Assume further that the lower bound of the constraint is increased by $\delta = d_{min} - c_{min}$. Theorem 5.1 then implies that the upper and lower bound of any other constraint in the new minimal time model will be restricted by at most δ as well. Thus, we are able to approximate the maximum difference between the new minimal time model and the original one. Similar rules apply to all other change operations (cf. Table 5.1) adding or restricting a temporal constraint.

From this, we can conclude that when applying another insert operation, it will be sufficient to verify that any precondition referring to a constraint $\langle [x, y]_{XY}, \beta \rangle$ of the minimal time model is satisfied for the respective approximated constraint $\langle [x+\delta, y-\delta]_{XY}, \beta \rangle$ as well. In this case the insert operation may be applied without violating the temporal consistency of the process instance. In particular, and this constitutes a fundamental advantage of the results presented by (LR14), we need not restore minimality of the adapted minimal time model prior to the application of the operation. By contrast, if the precondition is not met for the approximated constraint, it might still be possible to apply the change without violating temporal consistency. However, in this case, minimality of the modified minimal time model must be first restored before deciding whether the change may be applied.

By contrast, when removing an explicit constraint from the time model, basically, this results in the possible relaxation of some implicit constraints. As discussed in the previous section, it is not possible to restore minimality of a modified time model after relaxing one of its constraints. This is due to the fact that one cannot easily determine which other constraints have to be relaxed and to what extend. However, relaxing a constraint only results in the relaxation of other constraints. Particularly, no existing constraint is restricted by this change. Thus, it is not necessary to restore minimality of the minimal time model after each delete operation. Instead it is sufficient to restore its minimality if the precondition of a subsequent change operations cannot be met. Particularly, in such a case, it becomes necessary to check whether the change operation indeed violates temporal consistency of the process instance or the current approximation of the minimal time model is too strict.

Based on these observations it becomes possible to apply a sequence of change operations to a process instance within a single transaction (e. g., to insert and/or delete multiple activities) without need to restore minimality of the minimal time model after each change. Example 5.1 illustrates this approach.

Example 5.1 (Applying multiple change operations)
Figure 5.2 depicts a process schema and the corresponding minimal time model[2] to which a series of three change operations ⓐ-ⓒ shall be applied. First, X having duration $[4, 9]$ shall be inserted between A and ANDsplit (Figure 5.2 ⓐ). This is possible without violating the temporal consistency of the process schema as the minimum duration of X is lower than the maximum time distance between A and ANDsplit (i. e., $4 \leq 7$). After performing the change, the value used for approximating the minimal time model becomes $\delta = 4 - 0 = 4$. Next, Y shall be inserted between B and C (Figure 5.2 ⓑ). Again this is possible since the minimum duration is lower than the approximated maximum time distance (i. e., $9 \leq 14 - \delta = 10$). We can now use the approximated time model to approximate the time model resulting after this change; i. e., we may simply assume that δ is increased to $\delta = 4 + (9 - 7) = 6$ after the change. Subsequently, inserting Z with duration $[5, 8]$ between D and ANDjoin (Figure 5.2 ⓒ) is not possible based on the approximated minimal time model as the precondition of the respective change operation cannot be met (i. e., $5 \not\leq 10 - \delta = 4$). Hence, minimality of the minimal time model must be restored (Figure 5.2 ⓓ). Afterwards, inserting Z becomes possible as for the new minimal time model the precondition of the operation is met. Finally, minimality of the last minimal time model must be restored (Figure 5.2 ⓔ) before continuing the execution of the process instance.

[2]To improve readability implicit constraints have been omitted from the time model.

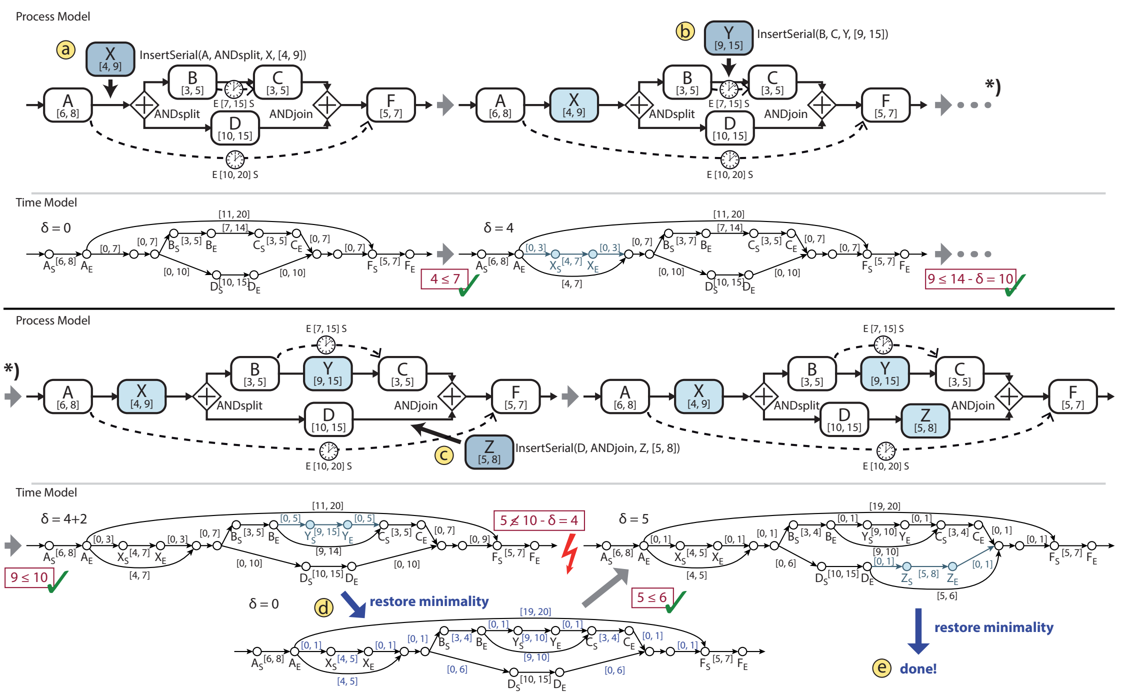


Figure 5.2: Applying multiple change operations to a process schema (LR14)

Proof-of-Concept Prototype

The approach presented by (LR14) was implemented as a proof-of-concept prototype as part of the ATAPIS Toolset [72]. This prototype allows users to apply the presented change operations to both process schemas and process instances. The implementation of the change operations is based on the well-founded set of change operations provided by the AristaFlow BPM Suite [34], which was used as basis for the ATAPIS Toolset. Overall the prototype demonstrates the applicability of the approach presented by (LR14). The prototype is shown in Figure 5.3: at the top, a process model from the healthcare domain comprising several temporal constraints is shown. At the bottom left, the automatically generated time model is depicted. At the bottom right, the corresponding minimal time model is shown. Finally, the right side displays the available set of change operations. Whether a particular change operation may be applied is decided by checking both structural and temporal preconditions. When applying the operation to the process instance all three models are updated simultaneously. Altogether the prototype allows us to efficiently provide the required flexibility for time-aware processes.

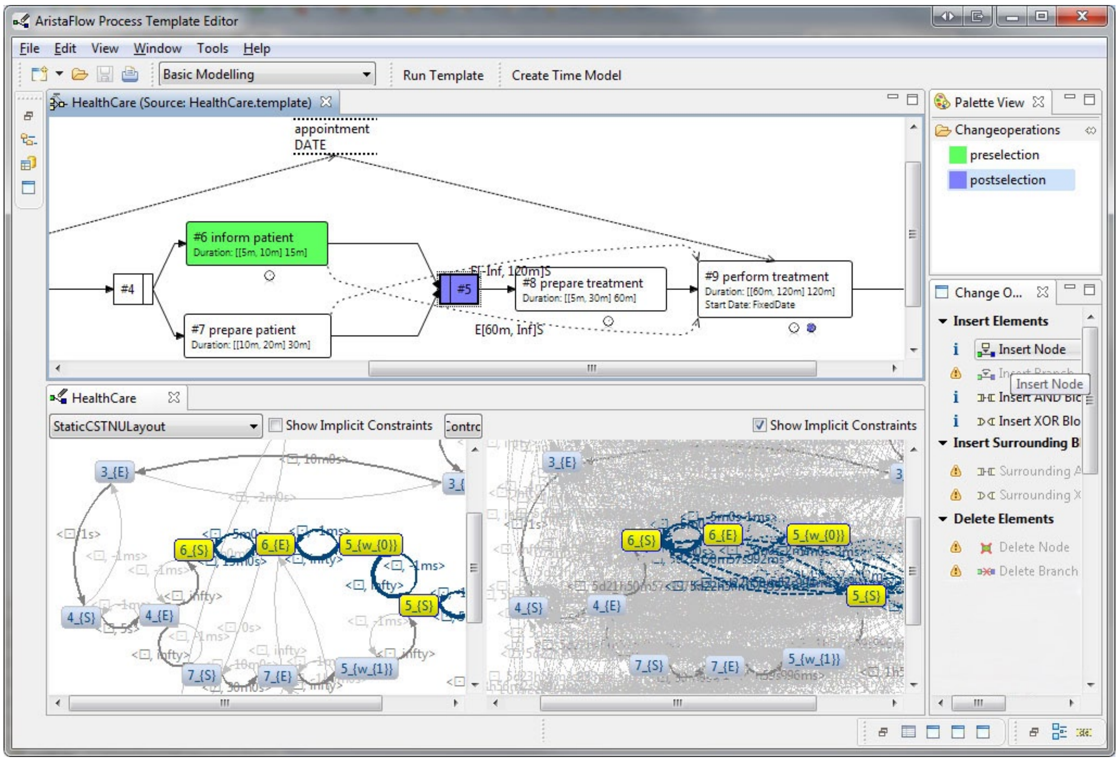


Figure 5.3: Screenshot of the ATAPIS Toolset with active change operations (LR14)

5.3 Evaluation and Related Work

Dynamic changes of processes without temporal constraints and adaptive PAIS were extensively studied in the past [18, 48, 121, 122, 124, 125, 130, 131, 156]. A comprehensive overview of the state of the art in adaptive PAIS and process flexibility is provided by [123].

Regarding dynamic process instance changes, the soundness of the modified process instance is crucial [18, 48, 128]. Besides the structural soundness of the underlying process schema, this encompasses the behavioral soundness of the modified process instance [18, 123, 130]. How to ensure behavioral soundness when applying a particular change operation has been studied in [130, 131, 156]. A widespread correctness notion for behavioral soundness is *state compliance* [18, 130] (cf. Section 2.3). Note that ensuring temporal consistency of the modified process schema, as discussed by (LR14), also allows ensuring state compliance of the time perspective; i. e., it ensures that the temporal execution trace is reproducible on the modified process schema.

Weber et al. [166] present 18 change patterns and 7 change support features frequently used for updating process schemas as well as process instances. In turn, [134] formally defines their semantics and—similar to the work presented by (LR14)—presents suitable pre- and post-conditions for each change pattern, which ensure that the resulting process instance remains structurally and behaviourally sound. In [165], a subset of the change patterns is used for refactoring process schemas in large process repositories, while preserving their behaviour.

In the context of process schema evolution challenging issues emerge as well. When establishing a new process schema version, for example, one has to decide whether already running process instances shall be migrated to the new process schema or be completed based on the previous schema version [123]. In this context, Rinderle et al. [129] discuss how to detect and deal with conflicts that may arise if an already modified process instance shall be migrated to a new process schema version.

To the best of our knowledge, the work by Sadiq et al. [143] is the only work considering dynamic changes in the context of time-aware processes. However, [143] only provides a high level discussion of the different aspects to be considered in this context. In particular, it proposes the use of a three-phase modification process for process evolution consisting of the steps definition of the modification, conforming the instances to be migrated to the modification, and enacting the modification. According to Sadiq et al., the management of temporal aspects constitutes an integral part of this process [143]. First, during the definition phase the modified process schema must be checked for temporal consistency. Then during the conforming phase the process instances to be migrated must brought into conformity with the new process schema. This includes re-ensuring the temporal consistency of the process instance. Finally, during the enactment phase the process instance must be monitored to prevent unforeseen violations of temporal constraints that may occur during the transition period. This three phase process is also valid for the migration of process instances when using the time-aware change operations proposed by (LR14). However, the complexity of the first two phases may be significantly reduced as the proposed change operations already allow ensuring the temporal consistency of the process schema and respective instances.

Different aspects related to dynamic changes have been studied in the context of temporal constraint networks as well. For example, Cesta and Oddi [20] investigate how the efficiency of the consistency checking of STN can be improved when adding or removing a temporal constraint to or from a minimal STN. In particular, they show that if a constraint is added to a minimal STN during the subsequent constraint propagation, each constraint may be updated at most once, otherwise the modified STN can no longer be consistent. Moreover, they show how a dependency tree between the temporal constraints of an STN can be used to decide which of the derived temporal constraints have to be reevaluated when removing a temporal constraint from the network. Finally, Planken et al. [113] present a more efficient consistency checking algorithm for STN based on the incremental insertion of temporal constraints. Both results can probably be extended to CSTN as well and be used to further increase the efficiency of the time-aware change operations proposed by (LR14).

5.4 Discussion and Outlook

In today's fast-paced world, where small delays and missed deadlines might have severe consequences, it is crucial for businesses to be able to control the temporal constraints of their business processes. As process execution does not always stick to the plan, it is further crucial that businesses are able to flexibly react to deviations in a time-aware process instance without jeopardizing its other properties, like soundness and temporal consistency. While soundness has been extensively studied in literature in the context of dynamic process changes, (LR14) is the first work considering temporal consistency in this context.

To this end, the referenced paper first defines a set of change operations for time-aware processes with suitable pre- and post-conditions. The latter ensure that, if a change operation is applied to a temporally consistent process instance, the changed process instance remains temporally consistent. Second, the referenced paper analyzes the effects, respective change operations have on the temporal constraints of the process instance. Based on the results it proposes a technique for approximating the resulting temporal properties of the changed process instance. This technique allows us to significantly reduce the complexity of the time calculations required when applying multiple change operations in the context of a single transaction.

In the future, we expect that the set of change operations presented by the referenced paper will be extended to cover more complex change patterns as well. In general, such change patterns can be realized based on a combination of change operations. However, dedicated implementations of the change patterns with more specific pre- and post-conditions might provide better support for respective change patterns. Moreover, we expect that the presented approach will be further investigated in the context of process evolution to evolve time-aware processes and migrate large sets of process instances to a new process schema. In this context, it will be particularly interesting to see how the algorithms can be further optimized and if clustering techniques can be used for separating different sets of process instances (e. g., easily migratable vs. questionable vs. not migratable) in order to further speed up the migration of large sets of process instances.

Part III

Conclusion

6 Summary and Outlook

All we have to decide is what to do with the time that is given us.

(Gandalf, Lord of the Rings)

Nowadays, being in control of its business processes is of utmost importance for any company. Many of these business processes are subject to temporal constraints which have to be observed during process execution (e. g., due to business rules and regulations or delivery contracts with business partners). To foster the management, execution, monitoring, and control of their business processes companies increasingly adopt Process-Aware Information Systems (PAISs) due to their promising perspectives for improved process support. Yet, although the proper support of the time perspective is crucial for many business processes, contemporary PAISs still lack a comprehensive support of the time perspective. This constitutes a severe limitation for the widespread use of PAISs in many application domains. Accordingly, the proper support of the time perspective has been ranked among the key challenges for further development and maturation of this promising technology [29, 33, 47, 120, 148].

To tackle some of the fundamental challenges emerging in this context this thesis contributes essential concepts and techniques for the wider-range integration of the time perspective in modern PAISs. Resulting *Adaptive Time- and Process-Aware Information System (ATAPIS)* will serve as an ideal environment for the systematic integration and automation of time-aware business processes.

In Section 6.1 we first summarize the contributions of this work with respect to time support in PAISs. Section 6.2 presents additional publications created during the author's Ph. D. project and shortly discusses their relationship with the contents of this thesis. Finally, Section 6.3 concludes the thesis with an outlook on future research challenges we identified during our research.

6.1 Contribution

ATAPIS Framework

This thesis contributes the *ATAPIS framework* enabling the integration of the time perspective as an integral part of the process life cycle (cf. Section 1.1) in adaptive PAISs. The ATAPIS framework enables the comprehensive analysis, description and formal specification of time-aware business processes by providing a set of well-founded and universal process time patterns. Moreover, it provides a formal basis and techniques for verifying the temporal consistency of resulting time-aware process schemas by first defining a formal semantics for each time pattern and subsequently proposing a technique for verifying the time perspective of a process schema using the aforementioned formal semantics. To support process enactment, a theoretical framework as well as an algorithm for monitoring and ensuring temporal consistency of time-aware processes during run time are presented. Finally, a set of change operations enabling the dynamic modification of time-aware process instances in a safe and sound manner is provided.

In summary:

Time Pattern

In Section 3.1, a set of 10 process *time patterns* comprising more than 100 different pattern variants is presented. Respective time patterns are based on empirical evidence and represent temporal concepts commonly occurring in real-world business processes. The time patterns provide a scientifically grounded, universal set of notions for describing temporal aspects in business processes. Hence, the time patterns facilitate the comparison of PAISs regarding their support of the time perspective and foster the informed selection of appropriate PAISs. Moreover, they may serve as a benchmark for assessing the support a particular process management technology provides for time-aware processes. The empirical evidence we gained in case studies, our evaluation of existing approaches, and the rapid pick up of the time patterns by other researchers have confirmed that the proposed time patterns are common in practice and are required for properly capturing the time perspective of business processes in many application domains. Therefore, they present an excellent and solid foundation for further investigating the support of time-aware processes in PAISs in a comprehensive way.

Precise Formal Semantics

To put the integration of the time patterns into PAISs on a sound and solid basis, Section 3.2 formally defines the *semantics* of each time pattern and its variants. To foster the use of the time patterns for a wide range of application scenarios this formal description of the time pattern semantics is specified in a language-independent manner based on temporal execution traces. This will facilitate the integration of the time patterns in a wide range of process modeling languages and process management tools as respective implementations do not have to cope with the specifics of another modeling language. The provision of the formal pattern semantics provides a key requirement for the development of advanced techniques which enable us to formally verify the time perspective of a (business) process at both design and run time as well as for properly supporting the enactment and monitoring of time-aware processes.

The formal semantics of the time patterns provide the foundation for supporting time-aware processes in PAISs. For this purpose, Section 4.1 proposes a framework for implementing the time patterns in PAISs, which is subsequently extended and refined in Section 4.2. The framework and corresponding techniques enable us to check whether a particular time-aware process schema is *temporally consistent* according to the defined formal pattern semantics. Moreover, it allows uncovering complex interdependencies between different temporal constraints, which otherwise might have gone unnoticed. This enables us to answer one of the fundamental questions related to the modeling and execution of time-aware processes, i. e., is it possible to successfully execute an instance of a process schema without violating any of its temporal constraints. The framework is subsequently extended to further consider the contingent, but restrictable nature of the duration of real-world activities as well as issues emerging in the context of alternative execution paths and run-time support. In particular, we provide a means to ensure that an instance of a time-aware process schema remains temporally consistent no matter how the durations of its activities actually turn out to be within their given bounds. Moreover, we ensure that a process instance my be successfully completed regardless of which path is chosen during run time and when respective decisions are made.

Temporal Consistency

At process *run time* actual activity durations become known, the values of some temporal constraint (e. g., appointments) are determined, and execution decisions are made. Hence, checking temporal consistency of time-aware process schemas only at design time is not sufficient. Therefore, Section 4.3 extends the proposed framework with necessary concepts and algorithms required for the execution and monitoring of time-aware process instances. In particular, we present an algorithm that enables flexible consistency checking of time-aware processes during run time, specifically considering the dynamic nature of certain temporal constraints. Based on the presented concepts it becomes possible to ensure that a time-aware process instance is executed without violating any temporal constraints. If, due to events not controllable by the PAIS, successful completion of a process instance is no longer possible, respective algorithms ensure that such situations may be detected as early as possible.

Run Time

To allow for the necessary flexibility during process execution, Chapter 5 contributes a set of *change operations* for dynamically adapting time-aware process instances. Respective change operations define suitable pre- and post-conditions to ensure that a modified time-aware process instance remains temporally consistent. Moreover, we present an approximation-based technique that allows significantly reducing the complexity of the time calculations required when applying multiple change operations in the context of a single transaction. This is particularly important in the context of process schema

Change Operations

evolution, where a potentially large set of process instances may have to be migrated to a new process schema version on-the-fly.

The ATAPIS framework and respective concepts have been implemented as a proof-of-concept prototype as part of the ATAPIS Toolset. The latter is based on the AristaFlow BPM Suite[1], a fully fledged process management system that provides advanced process support features [32, 74, 75, 123]. The ATAPIS Toolset allows specifying process schemas enriched with temporal constraints (cf. Section 3.1), which may then be checked for temporal consistency based on the presented framework (cf. Sections 4.1 and 4.2). Moreover, it can be used to simulate the execution of a time-aware process instance, including the possibility to check for constraint violations during run time (cf. Section 4.3). This prototype demonstrates the realizability as well as practical usability of the concepts presented in this thesis.

Altogether, the introduced concepts, techniques, and algorithms provide a significant contribution to the development of *Adaptive Time- and Process-Aware Information Systems*. Moreover, they will foster the widespread use of PAISs as they enable the usage of the latter in a broader set of application areas.

6.2 Additional Publications

In addition to the publications directly contributing to this thesis, the author of this thesis has been involved in number of other publications during his Ph. D. project, which are related to time and processes.

In [81, 83], we propose an extension of STNU, the Simple Temporal Network with Partially Shrinkable Uncertainty (STNPSU), which allows for the definition and efficient management of guarded links. A guarded link represents a generalization of contingent constraints (cf. Section 4.2). In particular, a guarded link represents an admissible range of delays between two timepoints, where each bound of the constraint may be restricted during run time, but not beyond a given threshold. Moreover, we present two algorithms for checking dynamic controllability of STNPSU and for executing dynamically controllable STNPSUs in a save way.

In [78], a user friendly visualization of the time perspective of time-aware processes based on enhanced Gantt charts is presented. Based on this, a method for creating personalized process schedules using process views [64, 65, 67] is suggested.

A method for generating optimized enactment plans from declarative process schemas with temporal constraints is suggested in [6]. The generated plans can be used to improve support of time-aware processes specified in a declarative way by providing users with personal schedules, predicting execution times of activities, and facilitating early detection of critical situations.

[1] www.aristaflow.com

Based on the results presented in [81, 83], in [84, 85] we present a method for representing and supporting modularized time-aware processes. In particular, we show how to derive the duration restriction of a time-aware (sub-)process in such a way that its temporal properties are completely specified. Moreover, we propose a novel approach for determining and representing the overall temporal behavior of a process, called *guarded range with contingency*. Using this representation, we can specify the possible durations of a (sub-)process as well as any permissible restriction that may be applied to it, while still ensuring the temporal consistency of the process. Finally, we show how this characterization of a process can be utilized when re-using it as a subprocess within a modularized process.

6.3 Outlook

Time support for PAISs is a wide research area, which can only be partially covered by one thesis. In particular, this thesis has unveiled several aspects that should be addressed by further research. Some of them were already mentioned in the discussion part of each chapter.

- *Time Patterns for other process perspectives.* The systematic literature review conducted as part of (LWR14) revealed several temporal constraints being relevant for other process perspectives. For example, [172, 173] emphasize the need of validity periods as well as maximum processing times for process data. Moreover, [104] indicates the need for temporally restricted process changes. Finally, many compliance rules refer to temporal aspects [94]. Such extended temporal constraints have been out of the scope of this thesis, which focuses on the basic support of time-aware processes. Moreover, for some of the extended temporal constraints empirical evidence is missing on whether they are required outside a specific application domain and thus represent a "pattern". Nevertheless, such extended time patterns should be investigated to provide a holistic support of the time perspective of business processes in PAISs.

- *Tool support for modeling time-aware process schemas.* So far, the ATAPIS Toolset only provides elementary support for modeling time-aware processes. In particular, domain experts with limited or no process modeling experience might consider it hard to specify required temporal constraints for a process schema. Hence, a more sophisticated modeling approach for time-aware processes is required, which specifically considers the more complex aspects of some of the time patterns (e. g., restrictable duration ranges, schedule restricted elements, periodicity). Moreover, practitioners should be encouraged to use respective tools and feedback given from them should be used for their further development. This involves the implementation and evaluation of user studies to provide proper method and tool support.

- *Comprehensive run-time support for time-aware processes.* The execution algorithm discussed in Section 4.3 enables the execution and monitoring of time-aware process instances in PAISs. However, there are still open issues remaining regarding the proper support of time-aware processes. For example, efficiency is a crucial aspect with respect to run-time support in PAISs. Particularly, if hundreds or thousands of process instances are executed concurrently this becomes increasingly important. In this context, for example, execution algorithms based on heuristics might proof beneficial. In particular, the use case analyzes performed as part of (LWR14; LRW16) revealed that in reality some temporal constraints (e. g., appointments) are more important and have greater impact on the temporal properties of a process instance than others. Moreover, some temporal constraints merely represent business rules or serve planning purposes and could thus be ignored if necessary, whilst the violation of other constraints will threaten the successful completion of a process instance altogether. If such information is made available in the process schema it might be used to provide better and more efficient run-time support for time-aware processes.
- *User integration.* User integration should be investigated more extensively; e. g., how can users be made aware of the temporal state of their process instances and how can the observance of temporal constraints be enforced with users. Moreover, despite all efforts process instances will not always stick to the plan. Therefore, proper exception handling strategies are required, which help users in handling temporal constraint violations.
- *Modularized time-aware processes.* Although, temporal constraints and process modularity seem to be orthogonal features that may be managed independently, when having a closer view it turns out that this is not the case. In particular, in order to support subprocesses in a true modular way, one needs to be able to represent the overall temporal properties of a time-aware process by a single node with a duration. However, due to the involved activities with contingent durations as well as alternative execution paths this is currently not fully possible.

Bibliography

[1] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*. Oxford University Press, 1977.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. In *Communications of the ACM*, volume 26, pages 832–843. ACM Press, 1983.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. DOI: 10.1016/0304-3975(94)90010-8.

[4] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1): 181–203, 1994. DOI: 10.1145/174644.174651.

[5] S. C. Bandinelli, A. Fuggetta, and C. Ghezzi. Software process model evolution in the spade environment. *IEEE Transactions on Software Engineering*, 19(12): 1128–1144, 1993. ISSN 0098-5589. DOI: 10.1109/32.249659.

[6] I. Barba, A. Lanz, B. Weber, M. Reichert, and C. del Valle. Optimized time management for declarative workflows. In *Proceedings of the 13th International Conference BPMDS 2012, 17th International Conference EMMSAD 2012, and 5th EuroSymposium*, volume 113 of *Lecture Notes in Business Information Processing*, pages 195–210. Springer, June 2012. DOI: 10.1007/978-3-642-31072-0_14.

[7] A. P. Barros, M. Dumas, and A. H. M. ter Hofstede. Service interaction patterns. In *Proceedings of the 3rd International Conference on Business Process Management (BPM'05)*, volume 3649 of *Lecture Notes in Computer Science*, pages 302–318. Springer, September 2005. DOI: 10.1007/11538394_20.

[8] J. Becker, M. Kugeler, and M. Rosemann, editors. *Process Management: A Guide for the Design of Business Processes.* Springer, 2003. DOI: 10.1007/978-3-540-24798-2.

[9] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In *Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007. DOI: 10.1007/978-3-540-73368-3_14.

[10] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273, 1991. DOI: 10.1109/32.75415.

[11] C. Bettini, X. S. Wang, and S. Jajodia. Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 11(3):269–306, 2002. DOI: 10.1023/A:1014048800604.

[12] C. Bettini, X. S. Wang, and S. Jajodia. Solving multi-granularity temporal constraint networks. *Artificial Intelligence*, 140(1-2):107–152, 2002. DOI: 10.1016/S0004-3702(02)00223-0.

[13] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. The expressive power of time petri nets. *Theoretical Computer Science*, 474:1–20, 2013. ISSN 0304-3975. DOI: 10.1016/j.tcs.2012.12.005.

[14] S. Buchwald, T. Bauer, and M. Reichert. Bridging the gap between business process models and service composition specifications. In *Service Life Cycle Tools and Technologies: Methods, Trends and Advances*, pages 124–153. IGI Global, November 2011. DOI: 10.4018/978-1-61350-159-7.ch007.

[15] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, Volume 1, A System of Patterns*, volume 1 of *Pattern-Oriented Software Architecture*. Wiley, 1995.

[16] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture, Volume 4, A Pattern Language for Distributed Computing*, volume 4 of *Pattern-Oriented Software Architecture*. Wiley, 2007.

[17] F. Buschmann, K. Henney, and D. C. Schmidt. *Pattern-Oriented Software Architecture, Volume 5, On Patterns and Pattern Languages*, volume 5 of *Pattern-Oriented Software Architecture*. Wiley, 2007.

[18] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data & Knowledge Engineering*, 24(3):211–238, 1998. DOI: 10.1016/S0169-023X(97)00033-5.

[19] A. Cerone and A. Maggiolo-Schettini. Time-based expressivity of time petri nets for system specification. *Theoretical Computer Science*, 216(1-2):1–53, 1999. ISSN 0304-3975. DOI: 10.1016/S0304-3975(98)00008-5.

[20] A. Cesta and A. Oddi. Gaining efficiency and flexibility in the simple temporal problem. In *Proceedings., Third International Workshop on Temporal Representation and Reasoning (TIME '96)*, pages 45–50, May 1996. DOI: 10.1109/TIME.1996.555676.

[21] F. Cicirelli, A. Furfaro, and L. Nigro. Actor-based simulation of PDEVS systems over HLA. In *41st Annual Simulation Symposium (ANSS'08)*, pages 229–236. IEEE, 2008. DOI: 10.1109/ANSS-41.2008.5.

[22] F. Cicirelli, A. Furfaro, and L. Nigro. Using time stream petri nets for workflow modelling analysis and enactment. *Simulation*, 89(1):68–86, 2013. DOI: 10.1177/0037549711434603.

[23] A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, and M. Roveri. Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In *Temporal Representation and Reasoning (TIME), 2014 21st International Symposium on*, pages 27–36. IEEE, Sept 2014. DOI: 10.1109/TIME.2014.21.

[24] C. Combi and G. Pozzi. Towards temporal information in workflow systems. In *Advanced Conceptual Modeling Techniques (ER'02 Workshops)*, volume 2784 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2003.

[25] C. Combi and G. Pozzi. Task scheduling for a temporal workflow management system. In *Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME'06)*, pages 61–68. IEEE, June 2006. DOI: 10.1109/TIME.2006.26.

[26] C. Combi and G. Pozzi. Temporalities for workflow management systems. In J. Cardoso and W. M. P. van der Aalst, editors, *Handbook of Research on Business Process Modeling*, chapter 12, pages 255–273. Idea Group Publishing, 2009.

[27] C. Combi, M. Gozzi, J. M. Juarez, B. Oliboni, and G. Pozzi. Conceptual modeling of temporal clinical workflows. In *Proceedings of the 14th International Symposium on Temporal Representation and Reasoning (TIME'07)*, pages 70–81. IEEE, June 2007. DOI: 10.1109/TIME.2007.45.

[28] C. Combi, M. Gambini, S. Migliorini, and R. Posenato. Modelling temporal, data-centric medical processes. In *Proceedings of the 2nd ACM SIGHIT symposium on International health informatics*, pages 141–150. ACM Press, 2012. DOI: 10.1145/2110363.2110382.

[29] C. Combi, M. Gozzi, R. Posenato, and G. Pozzi. Conceptual modeling of flexible temporal workflows. *ACM Transactions on Autonomous and Adaptive Systems*, 7 (2):19:1–19:29, 2012. ISSN 1556-4665. DOI: 10.1145/2240166.2240169.

[30] C. Combi, L. Hunsberger, and R. Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. In *Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART-2013)*, pages 144–156. SciTePress Digital Library, February 2013.

[31] C. Combi, L. Hunsberger, and R. Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited. In *Agents and Artificial Intelligence: 5th International Conference, ICAART 2013 - Revised Selected Papers*, volume 449 of *Communications in Computer and Information Science*, pages 314–331. Springer, 2014. DOI: 10.1007/978-3-662-44440-5_19.

[32] P. Dadam and M. Reichert. The ADEPT project: A decade of research and development for robust and flexible process support - challenges and achievements.

Computer Science - Research and Development, 22(2):81–97, 2009. DOI: 10.1007/s00450-009-0068-6.

[33] P. Dadam, M. Reichert, and K. Kuhn. Clinical workflows - the killer application for process-oriented information systems. In *Proceedings of the 4th International Conference on Business Information Systems (BIS'00)*, pages 36–59, 2000.

[34] P. Dadam, M. Reichert, S. Rinderle-Ma, A. Lanz, R. Pryss, M. Predeschly, J. Kolb, L. T. Ly, M. Jurisch, U. Kreher, and K. Goeser. From ADEPT to AristaFlow BPM Suite: A research vision has become reality. In *Proceedings Business Process Management (BPM'09) Workshops, 1st Int'l. Workshop on Empirical Research in Business Process Management (ER-BPM '09)*, volume 43 of *Lecture Notes in Business Information Processing*, pages 529–531. Springer, September 2009. DOI: 10.1007/978-3-642-12186-9_50.

[35] A. David, M. O. Möller, and W. Yi. Formal verification of uml statecharts with real-time extensions. In *Proceedings Fundamental Approaches to Software Engineering (FASE'02)*, volume 2306 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2002. DOI: 10.1007/3-540-45923-5_15.

[36] I. Davies, P. Green, M. Rosemann, M. Indulska, and S. Gallo. How do practitioners use conceptual modeling in practice? *Data & Knowledge Engineering*, 58(3): 358–380, 2006. ISSN 0169-023X. DOI: 10.1016/j.datak.2005.07.007.

[37] E. de Maria, A. Montanari, and M. Zantoni. An automaton-based approach to the verification of timed workflow schemas. In *Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME'06)*, pages 87–94. IEEE, June 2006. DOI: 10.1109/TIME.2006.6.

[38] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[39] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991. DOI: 10.1016/0004-3702(91)90006-6.

[40] M. Döhring, B. Zimmermann, and L. Karg. Flexible workflows at design- and runtime using bpmn2 adaptation patterns. In W. Abramowicz, editor, *Business Information Systems*, volume 87 of *Lecture Notes in Business Information Processing*, pages 25–36. Springer, 2011. DOI: 10.1007/978-3-642-21863-7_3.

[41] M. Diaz and P. Sénac. Time stream petri nets a model for timed multimedia information. In *Application and Theory of Petri Nets 1994*, volume 815 of *Lecture Notes in Computer Science*, pages 219–238. Springer, 1994. DOI: 10.1007/3-540-58152-9_13.

[42] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, editors. *Process-aware information systems*. Wiley-Interscience, 2005. ISBN 0471663069.

[43] J. Eder and E. Panagos. Managing time in workflow systems. In L. Fischer, editor, *Workflow Handbook 2001*, pages 109–132. Future Strategies Inc., 2000.

[44] J. Eder, P. Euthimios, H. Pozewaunig, and M. Rabinovich. Time management in workflow systems. In *Proceedings of the 3rd International Conference on Business Information Systems (BIS'99)*, pages 265–280. Springer, 1999.

[45] J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99)*, volume 1626 of *Lecture Notes in Computer Science*, pages 286–300. Springer, June 1999. DOI: `10.1007/3-540-48738-7`.

[46] J. Eder, W. Gruber, and E. Panagos. Temporal modeling of workflows with conditional execution paths. In *Proceedings of the 11th International Conference on Database and Expert Systems Applications (DEXA'00)*, volume 1873 of *Lecture Notes in Computer Science*, pages 243–253. Springer, September 2000.

[47] J. Eder, E. Panagos, and M. Rabinovich. Workflow time management revisited. In J. Bubenko, J. Krogstie, O. Pastor, B. Pernici, C. Rolland, and A. Sølvberg, editors, *Seminal Contributions to Information Systems Engineering*, pages 207–213. Springer, 2013. ISBN 978-3-642-36925-4. DOI: `10.1007/978-3-642-36926-1_16`.

[48] C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proceedings of Conference on Organizational Computing Systems (COCS'95)*, pages 10–21. ACM, 1995. DOI: `10.1145/224019.224021`.

[49] M. Falda, F. Rossi, and K. B. Venable. Strong, weak, and dynamic consistency in fuzzy conditional temporal problems. In *Proceedings of COPLAS 2007, CP'07 workshop on planning and scheduling*, 2007.

[50] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[51] T. A. Henzinger. It's about time: Real-time logics reviewed. In D. Sangiorgi and R. de Simone, editors, *Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 1998. ISBN 978-3-540-64896-3. DOI: `10.1007/BFb0055640`.

[52] T. A. Henzinger, J.-F. Raskin, and P. Y. Schobbens. The regular real-time languages. In K. Larsen, S. Skyum, and G. Winskel, editors, *Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 1998. ISBN 978-3-540-64781-2. DOI: `10.1007/BFb0055086`.

[53] A. R. Hevner and S. Chatterjee. *Design Research in Information Systems*, volume 22 of *Integrated Series in Information Systems*. Springer, 2010. DOI: `10.1007/978-1-4419-5653-8`.

[54] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[55] L. Hunsberger. A fast incremental algorithm for managing the execution of dynamically controllable temporal networks. In *Proceedings of the 17th International Symposium on Temporal Representation and Reasoning (TIME'10)*, pages 121–128. IEEE, September 2010. DOI: 10.1109/TIME.2010.16.

[56] L. Hunsberger. A faster execution algorithm for dynamically controllable stnus. In *Proceedings of the 20th International Symposium on Temporal Representation and Reasoning (TIME-2013)*, pages 26–33. IEEE, 2013.

[57] L. Hunsberger, R. Posenato, and C. Combi. The dynamic controllability of conditional STNs with uncertainty. In *Proceedings of the Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx)*, pages 21–28, 2012.

[58] L. Hunsberger, R. Posenato, and C. Combi. A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In *Temporal Representation and Reasoning (TIME), 2015 22nd International Symposium on*, pages 4–18, Sept 2015. DOI: 10.1109/TIME.2015.26.

[59] M. Huth and M. Ryan. *Logic in Computer Science - modelling and reasoning about systems*. Cambridge University Press, 2004.

[60] *IBM ILOG CPLEX V12.1 - User's Manual for CPLEX*. IBM Corporation, 2009. URL ftp://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf. (last accessed: 2016-04-25).

[61] S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press, 1996. ISBN 1850322228.

[62] M. Kircher and P. Jain. *Pattern-Oriented Software Architecture, Volume 3, Patterns for Resource Management*, volume 3 of *Pattern-Oriented Software Architecture*. Wiley, 2004.

[63] B. A. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, University of Durham, 2007.

[64] J. Kolb. *Abstraction, Visualization, and Evolution of Process Models*. Ph.D. thesis, Ulm University, 2015.

[65] J. Kolb and M. Reichert. A flexible approach for abstracting and personalizing large business process models. *Applied Computing Review*, 13(1):6–17, 2013. DOI: 10.1145/2460136.2460137.

[66] J. Kolb, P. Hübner, and M. Reichert. Automatically generating and updating user interface components in process-aware information systems. In *20th International Conference on Cooperative Information Systems*, number 7565 in Lecture Notes in Computer Science, pages 444–454. Springer, September 2012.

[67] J. Kolb, K. Kammerer, and M. Reichert. Updatable process views for user-centered adaption of large process models. In *Proceedings of the 10th International Conference on Service Oriented Computing (ICSOC'12)*, number 7636 in Lecture Notes in Computer Science, pages 484–498. Springer, October 2012. DOI: 10.1007/978-3-642-34321-6_32.

[68] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. ISSN 0922-6443. DOI: 10.1007/BF01995674.

[69] A. Kumar, S. R. Sabbella, and R. R. Barton. Managing controlled violation of temporal process constraints. In *Business Process Management: 13th International Conference, BPM 2015*, volume 9253 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2015. DOI: 10.1007/978-3-319-23063-4_20.

[70] A. Lanz and M. Reichert. Process change operations for time-aware processes. Technical Report UIB-2014-01, University of Ulm, 2014. URL http://dbis.eprints.uni-ulm.de/1027/.

[71] A. Lanz and M. Reichert. Dealing with changes of time-aware processes. In *Business Process Management — Proceedings of the 12th International Conference on Business Process Management (BPM'14)*, volume 8659 of *Lecture Notes in Computer Science*, pages 217–233. Springer, September 2014. DOI: 10.1007/978-3-319-10172-9_14.

[72] A. Lanz and M. Reichert. Enabling time-aware process support with the ATAPIS toolset. In *Proceedings of the BPM Demo Sessions 2014*, volume 1295 of *CEUR Workshop Proceedings*, pages 41–45. CEUR, 2014.

[73] A. Lanz, B. Weber, and M. Reichert. Time patterns in process-aware information systems - a pattern-based analysis - revised version. Technical Report UIB-2009-05, University of Ulm, Germany, 2009. URL http://dbis.eprints.uni-ulm.de/648/.

[74] A. Lanz, U. Kreher, M. Reichert, and P. Dadam. Enabling process support for advanced applications with the AristaFlow BPM Suite. In *Proceedings of the Business Process Management 2010 Demonstration Track*, number 615 in CEUR Workshop Proceedings, September 2010.

[75] A. Lanz, M. Reichert, and P. Dadam. Making business process implementations flexible and robust: Error handling in the AristaFlow BPM Suite. In *Proceedings of the CAiSE'10 Forum - Information Systems Evolution*, volume 592 of *CEUR Workshop Proceedings*, pages 18–25. CEUR, 2010.

[76] A. Lanz, M. Reichert, and P. Dadam. Robust and flexible error handling in the AristaFlow BPM Suite. In *Proc. CAiSE'10 Forum, Information Systems Evolution*, number 72 in Lecture Notes in Business Information Processing, pages 174–189. Springer, June 2010. DOI: 10.1007/978-3-642-17722-4_13.

[77] A. Lanz, B. Weber, and M. Reichert. Workflow time patterns for process-aware information systems. In *Enterprise, Business-Process and Information Systems Modeling — Proceedings of the 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010, held at CAiSE 2010*, volume 50 of *Lecture Notes in Business Information Processing*, pages 94–107. Springer, June 2010. DOI: 10.1007/978-3-642-13051-9_9.

[78] A. Lanz, J. Kolb, and M. Reichert. Enabling personalized process schedules with time-aware process views. In *Advanced Information Systems Engineering Workshops (CAiSE'13 Workshops)*, volume 148 of *Lecture Notes in Business Information Processing*, pages 205–216. Springer, 2013. DOI: 10.1007/978-3-642-38490-5_20.

[79] A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controllability of time-aware processes at run time. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences — Proceedings of the 21st International Conference on Cooperative Information Systems (CoopIS'13)*, number 8185 in Lecture Notes in Computer Science, pages 39–56. Springer, September 2013. DOI: 10.1007/978-3-642-41030-7_4.

[80] A. Lanz, M. Reichert, and B. Weber. A formal semantics of time patterns for process-aware information systems. Technical Report UIB-2013-02, University of Ulm, 2013. URL http://dbis.eprints.uni-ulm.de/894/.

[81] A. Lanz, R. Posenato, C. Combi, and M. Reichert. Simple temporal networks with partially shrinkable uncertainty (extended version). Technical Report UIB-2014-05, Ulm University, 2014. URL http://dbis.eprints.uni-ulm.de/1124/.

[82] A. Lanz, B. Weber, and M. Reichert. Time patterns for process-aware information systems. *Requirements Engineering*, 19(2):113–141, 2014. DOI: 10.1007/s00766-012-0162-3.

[83] A. Lanz, R. Posenato, C. Combi, and M. Reichert. Simple temporal networks with partially shrinkable uncertainty. In *Proceedings of the 7th International Conference on Agents and Artificial Intelligence (ICAART'15)*, pages 370–381. SCITEPRESS, January 2015. DOI: 10.5220/0005200903700381.

[84] A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controlling time-awareness in modularized processes (extended version). Technical Report UIB-2015-01, University of Ulm, 2015. URL http://dbis.eprints.uni-ulm.de/1133/.

[85] A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controlling time-awareness in modularized processes. In *Proceedings of the 17th International Conference BPMDS 2016*, pages 157–172. Springer, 2016. DOI: 10.1007/978-3-319-39429-9_11.

[86] A. Lanz, M. Reichert, and B. Weber. Process time patterns: A formal foundation. *Information Systems*, 57:28–68, 2016. DOI: 10.1016/j.is.2015.10.002.

[87] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, 1997. DOI: 10.1007/s100090050010.

[88] J. Lenhard, A. Schönberger, and G. Wirtz. Edit distance-based pattern support assessment of orchestration languages. In R. Meersman, T. Dillon, P. Herrero, A. Kumar, M. Reichert, L. Qing, B.-C. Ooi, E. Damiani, D. Schmidt, J. White, M. Hauswirth, P. Hitzler, and M. Mohania, editors, *On the Move to Meaningful Internet Systems: OTM 2011*, volume 7044 of *Lecture Notes in Computer Science*, pages 137–154. Springer, 2011. DOI: 10.1007/978-3-642-25109-2_10.

[89] J. Lenhard, A. Schönberger, and G. Wirtz. Streamlining pattern support assessment for service composition languages. In D. Eichhorn, A. Koschmider, and H. Zhang, editors, *Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS'11)*, volume 705 of *CEUR Workshop Proceedings*, pages 112–119. CEUR, 2011.

[90] R. Lenz and M. Reichert. IT support for healthcare processes - premises, challenges, perspectives. *Data & Knowledge Engineering*, 61(1):39–58, 2007. DOI: 10.1016/j.datak.2006.04.007.

[91] H. Li and Y. Yang. Dynamic checking of temporal constraints for concurrent workflows. *Electronic Commerce Research and Applications*, 4(2):124–142, 2005.

[92] H. Li, Y. Yang, and T. Chen. Resource constraints analysis of workflow specifications. *Journal of Systems and Software*, 73(2):271–285, 2004.

[93] M. Lohrmann. *Business Process Quality Management*. Ph.D. thesis, Ulm University, June 2015.

[94] L. T. Ly. *SeaFlows – A Compliance Checking Framework for Supporting the Process Lifecycle*. Ph.D. thesis, Ulm University, May 2013.

[95] L. T. Ly, S. Rinderle-Ma, D. Knuplesch, and P. Dadam. Monitoring business process compliance using compliance rule graphs. In *On the Move to Meaningful Internet Systems: OTM 2011 (CoopIS'11)*, volume 7044 of *Lecture Notes in Computer Science*, pages 82–99. Springer, 2011. DOI: 10.1007/978-3-642-25109-2_7.

[96] F. M. Maggi and M. Westergaard. Using timed automata for a priori warnings and planning for timed declarative process models. *International Journal of Cooperative Information Systems*, 23(01):1440003, 2014. DOI: 10.1142/S0218843014400036.

[97] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS 95*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995. DOI: 10.1007/3-540-59042-0_76.

[98] R. S. Mans, N. C. Russell, W. M. P. van der Aalst, A. J. Moleman, and P. J. M. Bakker. Schedule-aware workflow management systems. In *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'09)*, volume 6550 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2009.

[99] R. S. Mans, W. M. P. van der Aalst, N. C. Russell, P. J. M. Bakker, and A. J. Moleman. Process-aware information system development for the healthcare domain-consistency, reliability, and effectiveness. In *Business Process Management Workshops*, volume 43:7 of *Lecture Notes in Business Information Processing*, pages 635–646. Springer, 2010.

[100] O. Marjanovic. Dynamic verification of temporal constraints in production workflows. In *Proceedings of the 11th Australasian Database Conference (ADC'00)*, volume 22:2 of *Australian Computer Science Communications*, pages 74–81, 2000. DOI: 10.1109/ADC.2000.819816.

[101] O. Marjanovic and M. E. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Systems*, 1(2): 157–192, 1999.

[102] D. Müller, J. Herbst, M. Hammori, and M. Reichert. IT support for release management processes in the automotive industry. In *Proceedings of the 4th International Conference on Business Process Management (BPM'06)*, volume 4102 of *Lecture Notes in Computer Science*, pages 368–377. Springer, September 2006. DOI: 10.1007/11841760.

[103] R. Müller and E. Rahm. Dealing with logical failures for collaborating workflows. In *Proceedings of the 4th International Conference Cooperative Information Systems (CoopIS'00)*, volume 1901 of *Lecture Notes in Computer Science*, pages 210–223, 2000.

[104] R. Müller, U. Greiner, and E. Rahm. AgentWork: a workflow system supporting rule-based workflow adaptation. *Data & Knowledge Engineering*, 51(2):223–256, 2004.

[105] P. Morris. A structural characterization of temporal dynamic controllability. In *Principles and Practice of Constraint Programming - CP 2006*, volume 4204 of *Lecture Notes in Computer Science*, pages 375–389. Springer, 2006. DOI: 10.1007/11889205_28.

[106] P. Morris and N. Muscettola. Execution of temporal plans with uncertainty. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 491–497. AAAI Press, 2000.

[107] P. Morris and N. Muscettola. Temporal dynamic controllability revisited. In *AAAI Conference on Artificial Intelligence*, pages 1193–1198. AAAI Press, July 2005.

[108] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *International Joint Conference on Artificial Intelligence*, volume 1, pages 494–502. Morgan Kaufmann Publishers Inc., 2001.

[109] M. Muehlen and J. Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE'07)*, volume 4495 of *Lecture Notes in Computer Science*, pages 465–479. Springer, June 2008. DOI: 10.1007/978-3-540-69534-9_35.

[110] Object Management Group. Business Process Model and Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0, 2011.

[111] M. Pesic, M. H. Schonenberg, N. Sidorova, and W. M. P. van der Aalst. Constraint-based workflow models: Change made easy. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94. Springer, 2007. DOI: 10.1007/978-3-540-76848-7_7.

[112] S. Philipose. *Operations Research - A Practical Approach.* Tata McGraw-Hill, 1986.

[113] L. Planken, M. de Weerdt, and N. Yorke-Smith. Incrementally solving stns by enforcing partial path consistency. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS'10)*, pages 129–136. AAAI Press, May 2010.

[114] H. Pozewaunig, J. Eder, and W. Liebhart. ePert: Extending PERT for workflow managment systems. In *Advances in Databases and Information Systems (ADBIS'97)*, pages 217–224, 1997.

[115] F. Puhlmann and M. Weske. Using the pi-calculus for formalizing workflow patterns. In *Proceedings of the 3rd International Conference on Business Process Management (BPM'05)*, volume 3649 of *Lecture Notes in Computer Science*, pages 153–168. Springer, September 2005. DOI: 10.1007/11538394.

[116] F. Puhlmann. Soundness verification of business processes specified in the pi-calculus. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, Part I*, volume 4803 of *Lecture Notes in Computer Science*, pages 6–23. Springer, 2007. DOI: 10.1007/978-3-540-76848-7_3.

[117] Y. Qu, C. Lin, and J. Wang. Linear temporal inference of workflow management systems based on timed petri nets models. In *Proceedings of the 1st International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS'02)*, volume 2480 of *Lecture Notes in Computer Science*, pages 30–44. Springer, 2002. DOI: 10.1007/3-540-45785-2_3.

[118] E. Ramezani Taghiabadi, D. Fahland, B. F. Dongen, and W. M. P. Aalst. Diagnostic information for compliance checking of temporal compliance requirements. In *Advanced Information Systems Engineering: 25th International Conference*, volume 7908 of *Lecture Notes in Computer Science*, pages 304–320. Springer, 2013. DOI: 10.1007/978-3-642-38709-8_20.

[119] M. Reichert. *Dynamische Ablaufänderungen in Workflow-Management-Systemen.* Ph.D. thesis, Ulm University, 2000.

[120] M. Reichert. The next wave of research in business process management. In *14th International Conference on Perspectives in Business Informatics Research (BIR 2015)*, August 2015.

[121] M. Reichert and T. Bauer. Supporting ad-hoc changes in distributed workflow management systems. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4803 of *Lecture Notes in Computer Science*, pages 150–168. Springer, 2007.

[122] M. Reichert and P. Dadam. ADEPTflex – supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[123] M. Reichert and B. Weber. *Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies.* Springer, 2012. DOI: 10.1007/978-3-642-30409-5.

[124] M. Reichert, C. Hensinger, and P. Dadam. Supporting adaptive workflows in advanced application environments. In *EDBT Workshop on Workflow Management Systems (in conjunction with EDBT'98 conference)*, pages 100–109, 1998.

[125] M. Reichert, P. Dadam, M. Jurisch, U. Kreher, K. Göser, and M. Lauer. Architectural design of flexible process management technology. In *Proceedings of the PRIMIUM Subconference at the Multikonferenz Wirtschaftsinformatik (MKWI)*, 2008.

[126] M. Reichert, P. Dadam, S. Rinderle-Ma, M. Jurisch, U. Kreher, and K. Göser. Architecural principles and components of adaptive process management technology. In A. Heinzl, P. Dadam, P. Lockemann, and S. Kirn, editors, *PRIMIUM - Process Innovation for Enterprise Software*, number P-151 in Lecture Notes in Informatics, pages 81–97. Koellen-Verlag, 2009.

[127] S. Rinderle. *Schema Evolution in Process Management Systems.* Ph.D. thesis, Ulm University, 2004.

[128] S. Rinderle, M. Reichert, and P. Dadam. Evaluation of correctness criteria for dynamic workflow changes. In *Proceedings of the 1st International Conference on Business Process Management (BPM'03)*, volume 2678 of *Lecture Notes in Computer Science*, pages 41–57. Springer, June 2003. DOI: 10.1007/3-540-44895-0.

[129] S. Rinderle, M. Reichert, and P. Dadam. On dealing with structural conflicts between process type and instance changes. In *Proceedings of the 2nd International Conference on Business Process Management (BPM'04)*, volume 3080 of *Lecture Notes in Computer Science*, pages 274–289. Springer, June 2004. DOI: 10.1007/b98280.

[130] S. Rinderle, M. Reichert, and P. Dadam. Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases*, 16(1):91–116, 2004.

[131] S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems: A survey. *Data & Knowledge Engineering*, 50(1):9–34, 2004.

[132] S. Rinderle, U. Kreher, M. Lauer, P. Dadam, and M. Reichert. On representing instance changes in adaptive process management systems. In *Proceedings of the First IEEE Workshop on Flexibility in Process-aware Information Systems (ProFlex'06)*, pages 297–304. IEEE, 2006. DOI: 10.1109/WETICE.2006.51.

[133] S. Rinderle-Ma, M. Reichert, and B. Weber. Relaxed compliance notions in adaptive process management systems. In *Proceedings of the 27th International Conference on Conceptual Modeling (ER'08)*, volume 5231 of *Lecture Notes in Computer Science*, pages 232–247. Springer, 2008. DOI: 10.1007/978-3-540-87877-3_18.

[134] S. Rinderle-Ma, M. Reichert, and B. Weber. On the formal semantics of change patterns in process-aware information systems. In *Proceedings of the 27th International Conference on Conceptual Modeling (ER'08)*, volume 5231 of *Lecture Notes in Computer Science*, pages 279–293. Springer, 2008. DOI: 10.1007/978-3-540-87877-3_21.

[135] A. Rogge-Solti and M. Weske. Prediction of business process durations using non-markovian stochastic petri nets. *Information Systems*, 54:1–14, 2015. ISSN 0306-4379. DOI: http://dx.doi.org/10.1016/j.is.2015.04.004.

[136] A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008. ISSN 0306-4379. DOI: 10.1016/j.is.2007.07.001.

[137] N. Russell, W. M. van der Aalst, and A. H. M. ter Hofstede. *Workflow Patterns – The Definitive Guide*. MIT Press, 2016.

[138] N. C. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow resource patterns. Technical report, Eindhoven University of Technology, 2004.

[139] N. C. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow data patterns: Identification, representation and tool support. In *Proceedings of the 24th International Conference on Conceptual Modeling (ER'05)*, volume 3716 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 2005. DOI: 10.1007/11568322_23.

[140] N. C. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. Workflow resource patterns: Identification, representation and tool support. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232. Springer, 2005. DOI: 10.1007/11431855_16.

[141] N. C. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Exception handling patterns in process-aware information systems. Technical report, BPMCenter.org, 2006.

[142] S. W. Sadiq and M. E. Orlowska. Dynamic modification of workflows. Technical report, Department of Computer Science and Electrical Engineering, University of Queensland, Brisbane, Australia, 1998.

[143] S. W. Sadiq, O. Marjanovic, and M. E. Orlowska. Managing change and time in dynamic workflow processes. *International Journal of Cooperative Information Systems*, 9(1-2):93–116, 2000. DOI: 10.1142/S0218843000000077.

[144] S. Sadiq, M. Orlowska, and W. Sadiq. Specification and validation of process constraints for flexible workflows. *Information Systems*, 30(5):349–378, 2005.

[145] M. Sanchez and J. Villalobos. Expanding and refining workflow time patterns. In *Computing Colombian Conference (8CCC), 2013 8th*, pages 1–6. IEEE, 2013. DOI: 10.1109/ColombianCC.2013.6637511.

[146] M. Sayal, F. Casati, U. Dayal, and M.-C. Shan. Business process cockpit. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. VLDB Endowment, 2002.

[147] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*, volume 2 of *Pattern-Oriented Software Architecture*. Wiley, 2000.

[148] S. Schulte, C. Janiesch, S. Venugopal, I. Weber, and P. Hoenisch. Elastic business process management: State of the art and open challenges for BPM in the cloud. *Future Generation Computer Systems*, 46:36–50, 2015. ISSN 0167-739X. DOI: 10.1016/j.future.2014.09.005.

[149] L. R. Shaffer, J. B. Ritter, and W. L. Meyer. *The Critical-path Method*. McGraw-Hill Education, 1965.

[150] N. Sidorova, C. Stahl, and N. Trčka. Workflow soundness revisited: Checking correctness in the presence of data while staying conceptual. In *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE'10)*, pages 530–544. Springer, 2010. DOI: 10.1007/978-3-642-13094-6_40.

[151] A. H. M. ter Hofstede, W. M. P. van der Aalst, M. Adams, and N. C. Russell, editors. *Modern Business Process Automation*. Springer, 2010. DOI: 10.1007/978-3-642-03121-2.

[152] L. Thom, M. Reichert, and C. Iochpe. Activity patterns in process-aware information systems: Basic concepts and empirical evidence. *International Journal of Business Process Integration and Management*, 4(2):93–110, 2009. DOI: 10.1504/IJBPIM.2009.027778.

[153] N. Trčka, W. M. P. Aalst, and N. Sidorova. Data-flow anti-patterns: Discovering data-flow errors in workflows. In *Proceedings 21st International Conference Advanced Information Systems Engineering (CAiSE'09)*, volume 5565 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 2009. DOI: 10.1007/978-3-642-02144-2_34.

[154] I. Tsamardinos, T. Vidal, and M. Pollack. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints*, 8(4):365–388, 2003. ISSN 1383-7133. DOI: 10.1023/A:1025894003623.

[155] W. M. P. van der Aalst. Verification of workflow nets. In *Application and Theory of Petri Nets (PETRINETS'97)*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer, 1997.

[156] W. M. P. van der Aalst. Exterminating the dynamic change bug: A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3):297–317, 2001. ISSN 1572-9419. DOI: 10.1023/A:1011409408711.

[157] W. M. P. van der Aalst and K. M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2004.

[158] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. DOI: 10.1023/A:1022883727209.

[159] W. M. P. van der Aalst, M. Rosemann, and M. Dumas. Deadline-based escalation in process-aware information systems. *Decision Support Systems*, 43(2):492–511, 2007. DOI: 10.1016/j.dss.2006.11.005.

[160] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23(2):99–113, 2009.

[161] W. M. P. van der Aalst, M. Pesic, and M. Song. Beyond process mining: From the past to present and future. In *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE'10)*, volume 6051 of *Lecture Notes in Computer Science*, pages 38–52. Springer, June 2010. DOI: 10.1007/978-3-642-13094-6.

[162] R. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37:229–327, 2001. DOI: 10.1007/s002360000041.

[163] T. Vidal and H. Fargier. Contingent durations in temporal csps: from consistency to controllabilities. In *Temporal Representation and Reasoning, 1997. (TIME '97), Proceedings., Fourth International Workshop on*, pages 78–85. IEEE, May 1997. DOI: 10.1109/TIME.1997.600786.

[164] T. Vidal and M. Ghallab. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proceedings 12th European Conference on Artificial Intelligence*, pages 48–54. John Wiley and Sons, Chichester, 1996.

[165] B. Weber and M. Reichert. Refactoring process models in large process repositories. In Z. Bellahsène and M. Léonard, editors, *Advanced Information Systems Engineering*, volume 5074 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2008. ISBN 978-3-540-69533-2. DOI: 10.1007/978-3-540-69534-9_9.

[166] B. Weber, A. Lanz, and M. Reichert. Time patterns for process-aware information systems: A pattern-based analysis. Technical report, University of Ulm, Germany, 2008. URL http://dbis.eprints.uni-ulm.de/527/.

[167] B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data & Knowledge Engineering*, 66(3):438–466, 2008. DOI: 10.1016/j.datak.2008.05.001.

[168] M. Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*. IEEE, 2001. DOI: 10.1109/HICSS.2001.927082.

[169] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

[170] J. Wondoh, G. Grossmann, and M. Stumptner. Utilising bitemporal information for business process contingency management. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW Frontiers, pages 45:1–45:10. ACM, 2016. DOI: 10.1145/2843043.2843045.

[171] M. Wynn, H. M. W. Verbeek, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. Business process verification – finally a reality! *Business Process Management Journal*, 15(1):74–92, 2009. DOI: 10.1108/14637150910931479.

[172] J. Xie, Y. Tang, Q. He, and N. Tang. Research of temporal workflow process and resource modeling. In *Proceedings of the 9th International Conference on Computer Supported Cooperative Work in Design*, volume 1, pages 530–534. IEEE, 2005.

[173] Y. Yu, Y. Tang, L. Liang, and Z.-s. Feng. Temporal extension of workflow meta-model and its application. In *Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design*, volume 2, pages 293–297. IEEE, 2004.

[174] H. Zhuge, H.-K. Pung, and T.-Y. Cheung. Timed workflow: Concept, model, and method. In *Proceedings of the 1st International Conference on Web Information Systems Engineering*, volume 1, pages 183–189. IEEE, 2000.

[175] H. Zhuge, T.-Y. Cheung, and H.-K. Pung. A timed workflow process model. *Journal of Systems and Software*, 55(3):231–243, 2001.

Index

Acronyms

ATAPIS	Adaptive Time- and Process-Aware Information System
BPMN	Business Process Modeling and Notation
CSP	Constraint Satisfaction Problem
CSTN	Conditional Simple Temporal Network
CSTNU	Conditional Simple Temporal Network with Uncertainty
EIS	Enterprise Information System
IT	Information Technology
PAIS	Process-Aware Information System
SESE	Single Entry Single Exit
STN	Simple Temporal Network
STNU	Simple Temporal Network with Uncertainty
STPU	Simple Temporal Problem under Uncertainty
STNPSU	Simple Temporal Network with Partially Shrinkable Uncertainty

Part IV

Appendix

A Publications

This appendix contains the publications which are part of this doctoral thesis.

A.1 Time Patterns for Process-Aware Information Systems (LWR14)

The following article appeared as

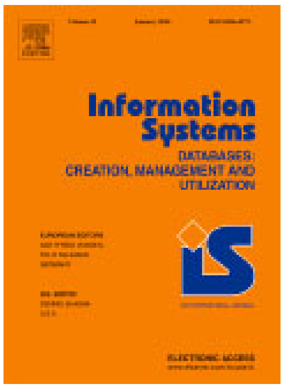
A. Lanz, B. Weber, and M. Reichert. Time patterns for process-aware information systems. *Requirements Engineering*, 19(2):113–141, 2014. DOI: 10.1007/s00766-012-0162-3

The original article is available at
http://link.springer.com/article/10.1007/s00766-012-0162-3

A.2 Process Time Patterns: A Formal Foundation (LRW16)

The following article appeared as

A. Lanz, M. Reichert, and B. Weber. Process time patterns: A formal foundation. *Information Systems*, 57:28–68, 2016. DOI: 10.1016/j.is.2015.10.002



The original article is available at
http://www.sciencedirect.com/science/article/pii/S0306437915300296

A.3 Controllability of time-aware processes at run time (LPCR13)

The following article appeared as

A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controllability of time-aware processes at run time. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences — Proceedings of the 21st International Conference on Cooperative Information Systems (CoopIS'13)*, number 8185 in Lecture Notes in Computer Science, pages 39–56. Springer, September 2013. DOI: 10.1007/978-3-642-41030-7_4

Robert Meersman Hervé Panetto
Tharam Dillon Johann Eder
Zohra Bellahsene Norbert Ritter
Pieter De Leenheer Deijing Dou (Eds.)

LNCS 8185

On the Move to
Meaningful Internet Systems:
OTM 2013 Conferences

Confederated International Conferences:
CoopIS, DOA-Trusted Cloud, and ODBASE 2013
Graz, Austria, September 2013, Proceedings

CoopIS
DOA TRUSTED cloud
ODBASE

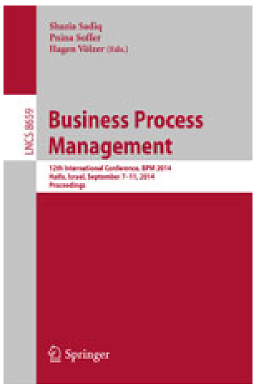
Springer

The original article is available at
http://link.springer.com/chapter/10.1007/978-3-642-41030-7_4

A.4 Dealing with changes of time-aware processes (LR14)

The following article appeared as

A. Lanz and M. Reichert. Dealing with changes of time-aware processes. In *Business Process Management — Proceedings of the 12th International Conference on Business Process Management (BPM'14)*, volume 8659 of *Lecture Notes in Computer Science*, pages 217–233. Springer, September 2014. DOI: 10.1007/978-3-319-10172-9_14



LNCS 8659

Shazia Sadiq
Pnina Soffer
Hagen Völzer (Eds.)

Business Process Management

12th International Conference, BPM 2014
Haifa, Israel, September 7-11, 2014
Proceedings

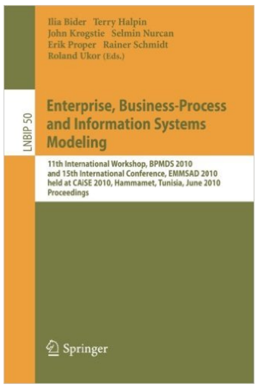
Springer

The original article is available at
http://link.springer.com/chapter/10.1007/978-3-319-10172-9_14

A.5 Workflow time patterns for process-aware information systems (LWR10)

The following article appeared as

A. Lanz, B. Weber, and M. Reichert. Workflow time patterns for process-aware information systems. In *Enterprise, Business-Process and Information Systems Modeling — Proceedings of the 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010, held at CAiSE 2010*, volume 50 of *Lecture Notes in Business Information Processing*, pages 94–107. Springer, June 2010. DOI: 10.1007/978-3-642-13051-9_9



Ilia Bider Terry Halpin
John Krogstie Selmin Nurcan
Erik Proper Rainer Schmidt
Roland Ukor (Eds.)

LNBIP 50

Enterprise, Business-Process and Information Systems Modeling

11th International Workshop, BPMDS 2010
and 15th International Conference, EMMSAD 2010
held at CAiSE 2010, Hammamet, Tunisia, June 2010
Proceedings

Springer

The original article is available at
http://link.springer.com/chapter/10.1007/978-3-642-13051-9_9

B Complete List of Publications

The following is the complete list of publications the author of this doctoral thesis has been involved in.

Peer-Reviewed Journals

- A. Lanz, B. Weber, and M. Reichert. Time patterns for process-aware information systems. *Requirements Engineering*, 19(2):113–141, 2014. DOI: 10.1007/s00766-012-0162-3

- A. Lanz, M. Reichert, and B. Weber. Process time patterns: A formal foundation. *Information Systems*, 57:28–68, 2016. DOI: 10.1016/j.is.2015.10.002

Peer-Reviewed Conferences

- I. Barba, A. Lanz, B. Weber, M. Reichert, and C. del Valle. Optimized time management for declarative workflows. In *Proceedings of the 13th International Conference BPMDS 2012, 17th International Conference EMMSAD 2012, and 5th EuroSymposium*, volume 113 of *Lecture Notes in Business Information Processing*, pages 195–210. Springer, June 2012. DOI: 10.1007/978-3-642-31072-0_14

- A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controllability of time-aware processes at run time. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences — Proceedings of the 21st International Conference on Cooperative Information Systems (CoopIS'13)*, number 8185 in Lecture Notes in Computer Science, pages 39–56. Springer, September 2013. DOI: 10.1007/978-3-642-41030-7_4

- A. Lanz and M. Reichert. Dealing with changes of time-aware processes. In *Business Process Management — Proceedings of the 12th International Conference on Business Process Management (BPM'14)*, volume 8659 of *Lecture Notes in Computer Science*, pages 217–233. Springer, September 2014. DOI: 10.1007/978-3-319-10172-9_14

- A. Lanz, R. Posenato, C. Combi, and M. Reichert. Simple temporal networks with partially shrinkable uncertainty. In *Proceedings of the 7th International Conference on Agents and Artificial Intelligence (ICAART'15)*, pages 370–381. SCITEPRESS, January 2015. DOI: 10.5220/0005200903700381

- A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controlling time-awareness in modularized processes. In *Proceedings of the 17th International Conference BPMDS 2016*, pages 157–172. Springer, 2016. DOI: 10.1007/978-3-319-39429-9_11

Peer-Reviewed Workshops

- A. Lanz, B. Weber, and M. Reichert. Workflow time patterns for process-aware information systems. In *Enterprise, Business-Process and Information Systems Modeling — Proceedings of the 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010, held at CAiSE 2010*, volume 50 of *Lecture Notes in Business Information Processing*, pages 94–107. Springer, June 2010. DOI: 10.1007/978-3-642-13051-9_9

- A. Lanz, J. Kolb, and M. Reichert. Enabling personalized process schedules with time-aware process views. In *Advanced Information Systems Engineering Workshops (CAiSE'13 Workshops)*, volume 148 of *Lecture Notes in Business Information Processing*, pages 205–216. Springer, 2013. DOI: 10.1007/978-3-642-38490-5_20

Demo Tracks

- A. Lanz, M. Reichert, and P. Dadam. Robust and flexible error handling in the AristaFlow BPM Suite. In *Proc. CAiSE'10 Forum, Information Systems Evolution*, number 72 in Lecture Notes in Business Information Processing, pages 174–189. Springer, June 2010. DOI: 10.1007/978-3-642-17722-4_13

- A. Lanz, M. Reichert, and P. Dadam. Making business process implementations flexible and robust: Error handling in the AristaFlow BPM Suite. In *Proceedings of the CAiSE'10 Forum - Information Systems Evolution*, volume 592 of *CEUR Workshop Proceedings*, pages 18–25. CEUR, 2010

- A. Lanz, U. Kreher, M. Reichert, and P. Dadam. Enabling process support for advanced applications with the AristaFlow BPM Suite. In *Proceedings of the Business Process Management 2010 Demonstration Track*, number 615 in CEUR Workshop Proceedings, September 2010

- A. Lanz and M. Reichert. Enabling time-aware process support with the ATAPIS toolset. In *Proceedings of the BPM Demo Sessions 2014*, volume 1295 of *CEUR Workshop Proceedings*, pages 41–45. CEUR, 2014

Technical Reports

- B. Weber, A. Lanz, and M. Reichert. Time patterns for process-aware information systems: A pattern-based analysis. Technical report, University of Ulm, Germany, 2008. URL http://dbis.eprints.uni-ulm.de/527/
- A. Lanz, B. Weber, and M. Reichert. Time patterns in process-aware information systems - a pattern-based analysis - revised version. Technical Report UIB-2009-05, University of Ulm, Germany, 2009. URL http://dbis.eprints.uni-ulm.de/648/
- A. Lanz, M. Reichert, and B. Weber. A formal semantics of time patterns for process-aware information systems. Technical Report UIB-2013-02, University of Ulm, 2013. URL http://dbis.eprints.uni-ulm.de/894/
- A. Lanz and M. Reichert. Process change operations for time-aware processes. Technical Report UIB-2014-01, University of Ulm, 2014. URL http://dbis.eprints.uni-ulm.de/1027/
- A. Lanz, R. Posenato, C. Combi, and M. Reichert. Simple temporal networks with partially shrinkable uncertainty (extended version). Technical Report UIB-2014-05, Ulm University, 2014. URL http://dbis.eprints.uni-ulm.de/1124/
- A. Lanz, R. Posenato, C. Combi, and M. Reichert. Controlling time-awareness in modularized processes (extended version). Technical Report UIB-2015-01, University of Ulm, 2015. URL http://dbis.eprints.uni-ulm.de/1133/

C Discussion of Personal Contribution

The following is a full list of all persons contributing to this dissertation.

Prof. Dr. Manfred Reichert	Universität Ulm Institut für Datenbanken und Informationssysteme James-Franck-Ring, Geb. O27/523 89081 Ulm (Germany)
Prof. Dr. Barbara Weber	Technical University of Denmark Department of Applied Mathematics and Computer Science Asmussens Allé, Building 303B 2800 Kgs. Lyngby (Denmark)
Prof. Roberto Posenato	Università degli Studi di Verona Dipartimento di Informatica Strada le Grazie 15 37134 Verona (Italy)
Prof. Carlo Combi	Università degli Studi di Verona Dipartimento di Informatica Strada le Grazie 15 37134 Verona (Italy)

Their contribution to the individual papers which are part of this thesis is discussed in the following.

C.1 Workflow time patterns for process-aware information systems (LWR10)

The author of this doctoral thesis was responsible for creating the initial list of candidate patterns and for analyzing the data sources to find empirical evidence for each of the time patterns. Moreover, he developed the proposed initial formal semantics and formulated the description of the presented time patterns. Prof. Weber advised the candidate in the development of a proper scientific methodology for the initial pattern identification and their validation. All of the authors (i. e., the candidate, Prof. Weber, and Prof. Reichert) were responsible for interpreting ambiguous cases found in the data sources and for the final classification of the identified concepts into the ten time patterns. Both, Prof. Weber and Prof. Reichert provided assistance in proof-reading the draft of the paper.

C.2 Time patterns for process-aware information systems (LWR14)

The author of this doctoral thesis was responsible for creating the initial list of candidate patterns and for analyzing the data sources to find empirical evidence for each of the time patterns. Moreover, he formulated the description of the presented time patterns. The candidate was further responsible for conducting the systematic literature review and for the evaluation of selected approaches from academia and industry. He also performed the classification of the time patterns with respect to their run-time characteristics. Prof. Weber advised the candidate in the development of a proper scientific methodology and in the implementation of a systematic literature review. All of the authors (i. e., the candidate, Prof. Weber, and Prof. Reichert) were responsible for interpreting ambiguous cases found in the data sources and for the final classification of the identified concepts into the ten time patterns. Both, Prof. Weber and Prof. Reichert provided assistance in proof-reading the draft of the paper.

C.3 Controllability of time-aware processes at run time (LPCR13)

The paper was prepared by the author of this doctoral thesis in close collaboration with Prof. Posenato and Prof. Combi. The author of the thesis was responsible for developing the presented conceptual model for specifying time-aware process schemas. Moreover, he developed and evaluated the concept of restrictable time intervals and contributed his insights on the dynamic nature of certain temporal constraints. The presented transformation of time-aware process schemas to CSTNU and the corresponding execution algorithm where developed in close collaboration by the three main authors (i. e., the candidate, Prof. Posenato, and Prof. Combi). The implementation of the presented approach in the ATAPIS Toolset was realized by the candidate. Prof. Reichert provided assistance in proof-reading the draft of the paper and advised the other authors in the suitable presentation of the results for the intended audience.

C.4 Dealing with changes of time-aware processes (LR14)

The paper was mostly prepared by the author of this doctoral thesis. The candidate was responsible for defining, analyzing and describing the proposed change operations. Moreover, he proposed the main theorem presented by the paper and subsequently showed its correctness. Prof. Reichert assisted the candidate by providing additional insights about process change operations and process change in general. Moreover, he assisted in proof-reading the draft of the paper and provided valuable comments which helped to improve the writing of the paper.

C.5 Process Time Patterns: A Formal Foundation (LRW16)

The author of this doctoral thesis was responsible for analyzing the data source and extracting the semantics of the time patterns. In this context, any unclear cases were discussed and resolved by all the authors. The candidate was further responsible for developing the formal description of the semantics of the time patterns. Moreover, he formulated the description of the presented time patterns semantics. The reference implementation of the time patterns was made by the candidate. All of the authors (i. e., the candidate, Prof. Reichert, and Prof. Weber) were responsible for identifying possible threats to validity and proper measures to deal with them. Prof. Weber advised the candidate in the development of a proper scientific methodology. Both, Prof. Reichert and Prof. Weber provided assistance in proof-reading the draft of the paper.