Process Time Patterns: A Formal Foundation

Andreas Lanz[a,*], Manfred Reichert[a], Barbara Weber[b]

[a] *Institute of Databases and Information Systems, Ulm University, Germany*
[b] *Quality Engineering Research Group, University of Innsbruck, Austria*

Abstract

Companies increasingly adopt process-aware information systems (PAISs) to model, execute, monitor, and evolve their business processes. Though the handling of temporal constraints (e.g., deadlines, or time lags between activities) is crucial for the proper support of business processes, existing PAISs vary significantly regarding the support of the temporal perspective. Both the formal specification and the operational support of temporal constraints constitute fundamental challenges in this context. In previous work, we introduced *process time patterns*, which facilitate the comparison and evaluation of PAISs in respect to their support of the temporal perspective. Furthermore, we provided empirical evidence for these time patterns. To avoid ambiguities and to ease the use as well as the implementation of the time patterns, this paper formally defines their semantics. To additionally foster the use of the patterns for a wide range of process modeling languages and to enable pattern integration with existing PAISs, the proposed semantics are expressed independently of a particular process meta model. Altogether, the presented pattern formalization will be fundamental for introducing the temporal perspective in PAISs.

Keywords: Process-aware Information System, Workflow Patterns, Process Time Patterns, Temporal Perspective, Temporal Constraints, Formal Semantics

1. Introduction

Companies strive for comprehensive life cycle support of their business processes [1, 2]. In particular, IT support for analyzing, modeling, executing, monitoring, and evolving business processes is becoming increasingly important [3, 4]. In this context, process-aware information systems (PAISs) offer promising perspectives by enabling companies to define their business processes in terms of explicit *process schemas* as well as to create, execute and monitor related *process instances* in a controlled and efficient manner [1].

Both the formal specification and the operational support of *temporal constraints* constitute fundamental challenges for PAISs [5, 6, 7, 8]. Although there exist many proposals for supporting the temporal process perspective, no comprehensive criteria for systematically assessing its support by a PAIS exist. To foster comparability and to facilitate the selection of PAIS-enabling technologies in a given application environment, *workflow patterns* have been introduced [9, 10, 11, 12]. Respective patterns allow analyzing the expressiveness of process modeling languages and tools in respect to different process perspectives, including control flow [9], data flow [10], resources [11], activities [13], exceptions [14], and process changes [12, 15, 16]. Recently, we extended the workflow patterns by a set of 10 *process time patterns* (*time patterns* for short) suitable for evaluating the support of the temporal perspective in PAISs [17, 18]. Examples of time patterns include *Time Lags between Activities*, *Durations*, and *Fixed Date Elements*. Empirical evidence, we gained in case studies [18], has confirmed that the proposed time patterns are common in practice and are required for properly modeling the temporal perspective of processes in many application domains [18]. Finally, we evaluated different approaches and tools in respect to their time pattern support [18].

*Corresponding author
Email addresses: andreas.lanz@uni-ulm.de (Andreas Lanz), manfred.reichert@uni-ulm.de (Manfred Reichert), barbara.weber@uibk.ac.at (Barbara Weber)

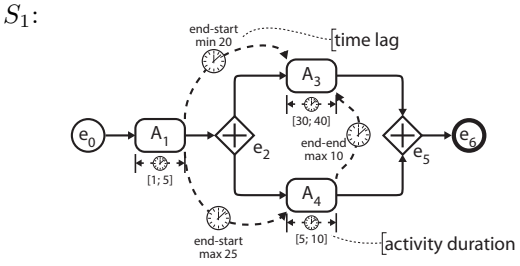

Figure 1: Interdependencies between different temporal constraints

1.1. Problem Statement

Our evaluation of approaches and tools incorporating the temporal process perspective has revealed the need for precise formal semantics of the time patterns. In particular, if such formal semantics are not present, patterns may be interpreted differently and ambiguities regarding informal pattern descriptions will result. In turn, this would hamper both pattern implementation and a pattern-based comparison of PAISs. Only when providing these precise formal semantics, it can be ensured that different implementations of a particular time pattern share the same semantics and, hence, have the same effects during process enactment. Precise formal semantics further constitute a prerequisite for verifying the temporal perspective of a business process at both build- and run-time [5, 6, 7, 19, 20], i.e., to check whether the temporal constraints of a process are satisfiable. Moreover, formal semantics are needed to be able to detect temporal inconsistencies in a process schema caused by interactions among different time pattern occurrences. Example 1 illustrates how such interactions may result in hidden effects. In particular, note that the occurrence of a time pattern within a process schema can never be treated in isolation. This significantly differentiates the time patterns from related patterns (e.g., workflow or data patterns [9, 10]), making a formal specification of their semantics indispensable. Only then a robust and error-free process execution becomes possible. Precise formal semantics are further required to achieve a common understanding of process schemas using the time patterns.

Example 1 (Interactions between temporal constraints). *Fig. 1 depicts process schema S_1 consisting of three activities and two control gateways. Each activity is associated with a minimum and maximum duration (Pattern: 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: Time Lags between Activities). At first glance, the process schema seems to be sound. However, when taking a closer look at it, one realizes that S_1 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., it 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 completes at most 35 times 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.*

In order to tackle the issues and limitations outlined above, formal semantics need to specify how the various time patterns interact with the elements of the control flow perspective, i.e., control flow patterns like loop, XOR-split, or AND-join. Moreover, in the context of loops and concurrent data access, for time patterns referring to process instance data (e.g., appointments made during run-time), it must be precisely defined which version of the data value shall be used for a specific pattern instance. For example, if a data object may be modified more than once during run-time, it must always be clear which version of the data value shall be used when evaluating a particular temporal constraint referring to the data object.

Since 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. A particular challenge is to provide a formal description of the time patterns, which is independent of a specific process modeling language or paradigm. Only then time patterns as well as their formal semantics will be widely accepted. Finally, this is required to enable PAIS engineers to integrate the time patterns without need to cope with language-specific issues.

1.2. Contribution

This paper complements our previous work on time patterns [17, 18] by providing *precise formal semantics* for them. These semantics are defined independent of a specific process modeling language or paradigm. Furthermore, we illustrate the pattern semantics through realistic examples and detailed explanations.

To define the pattern semantics independent of any process modeling language, while still closely intertwined with process execution semantics, we use *execution traces* as the basis for the formalization [21]. An execution trace may be considered as logical representation of the *execution history* of a process instance, i.e., it reflects what happened during the execution of a particular process instance. We formally describe time pattern semantics by characterizing which traces are producible on a process schema that contains a particular time pattern, i.e., we formally describe which traces are *temporally compliant* with the pattern semantics. This enables the implementation of techniques for checking the conformance of a process instance [22] with respect to a given process schema and its temporal constraints (i.e., occurrences of the time patterns). For pattern occurrences depending on run-time data of the process instance (i.e., on process instance data), we define which version of a data value shall be valid for a specific pattern instance. Finally, based on the presented formal semantics, we provide a reference implementation of selected time pattern variants.

As will be shown, the provided formal semantics contribute to overcome the problems discussed in Sect. 1.1. In particular, they will foster the integration of the temporal perspective into PAISs broadening their application scope significantly.

Sect. 2 provides background information and summarizes the time patterns. Sect. 3 discusses the research method we applied for defining and evaluating the proposed formal semantics. Sect. 4 then provides a formal semantics for each of the 10 time patterns. Sect. 5 elaborates on our reference implementation of the time patterns. In Sect. 6, we discuss the expected impact of the proposed formal pattern semantics as well as their usefulness for implementing time support in PAISs. Sect. 7 deals with potential threats regarding the validity of our work. Sect. 8 discusses related work and Sect. 9 concludes with a summary.

2. Backgrounds

This section provides basic notions needed for understanding this paper. Further, it summarizes the time patterns we informally introduced in [17, 18].

2.1. Basic Notions

A *process-aware information system* (PAIS) is a specific type of information system, providing process support functions and separating process logic from application code. At build-time, the process logic is explicitly defined based on the elements provided by a *process meta model* (cf. Fig. 2). For each business process to be supported by the PAIS, a *process type* represented by a *process schema* is defined. The latter corresponds to a directed graph that comprises a set of *nodes*, representing *activities* and *gateways* (e.g., Start-/End-Nodes and AND-joins), as well as a set of *control edges* specifying precedence relations between nodes (cf. Fig. 2). Furthermore, the notion of *activity set* refers to any subset of the activities of a process schema, whereby the elements of an activity set do not have to meet any structural requirements (e.g., connected component). When referring to specific regions of a process schema, in turn, we use the notion of *process fragment*. The latter refers to a sub-graph of a process schema with a single entry and a single exit node. 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. Fig. 2). More precisely, a data edge either represents a read or a write access of the respective activity to the referred data object.

At run-time, *process instances* may be created and executed according to the predefined process schema. An *activity instance*, in turn, represents the execution of a single activity in the context of a particular process instance. During run-time, a specific data element may be written several times, resulting in different *versions of a data value*. Finally, a *process instance set* refers to a collection of process instances executed by the PAIS. When executing a process instance, *events* may be triggered either by the process instance itself (e.g., start of the process instance), a process node (e.g., start of an activity instance, cf. Fig. 3), or an external source (e.g., receipt of a message). We use the notion of *event* as a general term for something

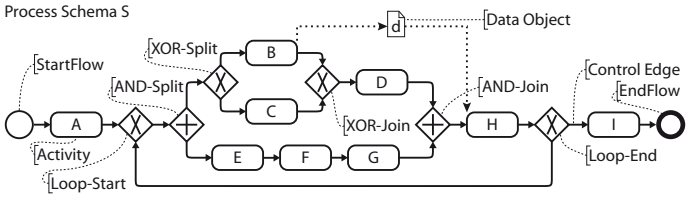

Figure 2: Core concepts of a process meta model

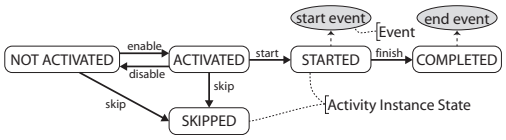

Figure 3: Activity life cycle and relevant events

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 1: Process time pattern catalogue [18]

happening during process execution. We assume that activity instances are executed according to the activity life cycle depicted in Fig. 3.When starting a process instance, its activities are in state *Not Activated*. As soon as a particular activity becomes enabled, it enters state *Activated*. When a user starts an activated activity instance it enters state *Started* and a corresponding *start event* is created. As soon as the user completes the activity instance, its state switches to *Completed* and an *end event* is generated. Finally, non-executed activities (e.g., activities in a non-selected path of an XOR-split) enter state *Skipped*.

2.2. Process Time Patterns

Process Time Patterns (TP) represent temporal constraints commonly occurring in PAISs (cf. Fig. 1) [17, 18]. The 10 time patterns can be divided into 4 distinct categories, based on their implications (cf. Fig. 1). *Pattern Category I* (*Durations and Time Lags*) provides support for expressing *durations* of process elements at different levels of granularity, i.e., activities, activity sets, processes, or sets of process instances. Furthermore, this category covers *time lags* between activities or—more generally—between process events (e.g., milestones). *Pattern Category II* (*Restricting Execution Times*) allows specifying constraints regarding possible execution times of single activities or an entire process (e.g., deadlines). *Category III* (*Variability*) provides support for expressing *time-based variability* during process execution (e.g., varying control-flow depending on temporal aspects). Finally, *Category IV* (*Recurrent Process Elements*) comprises patterns for expressing temporal constraints in connection with recurrent activities or process fragments (e.g., cyclic flows and periodicity). Example 2 introduces a simple process along which we will illustrate selected time patterns. A basic understanding of these patterns is required to understand the formalization provided in Sect. 4.

Example 2 (Patient Treatment Process). *Consider the simplified patient treatment process depicted in Fig. 4.*[1] *First, a doctor orders a medical procedure for a patient. Then, the responsible nurse makes an appointment with the medical department the treatment will take place in. Before that, the patient needs to be informed about the procedure and be prepared for it. Just before the treatment starts, a specific preparation is required. In turn, after completing the treatment, the doctor creates a short report (if requested). Finally, aftercare is provided and the doctor creates a final medical report, which is then added to the patient record.*

When considering the temporal perspective of this business process, a number of temporal constraints can be observed. Example 3 relates these temporal constraints to the time patterns of Fig. 1.

[1]Note that we use an extension of BPMN to visualize time patterns in process schemas (see [18] for details).

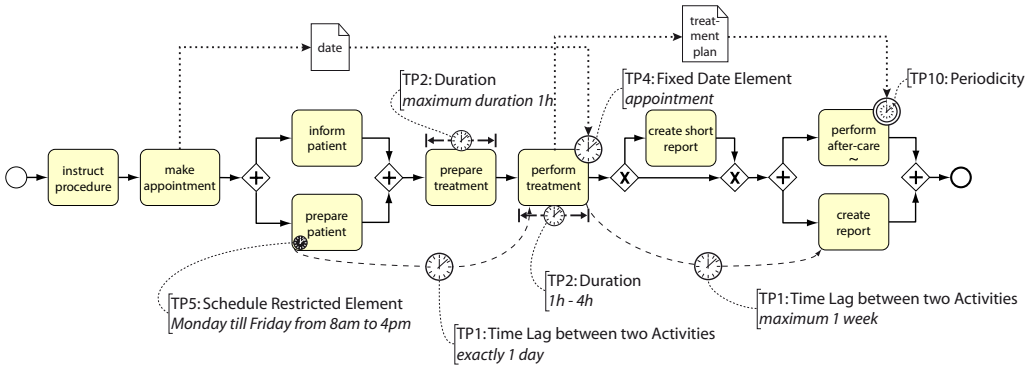

Figure 4: Simplified treatment process with temporal constraints

Example 3 (Temporal Constraints). *Reconsider Fig. 4. The* appointment *of the* `treatment`*, made in the context of activity* `make appointment`*, must be met during process execution. This constitutes a* ***Fixed Date Element*** *(TP4), restricting the earliest start date of activity* `perform treatment`*. In particular, the respective date will be set during run-time by activity* `make appointment` *and be stored in data object* `date`*. Moreover, this constraint affects the scheduling of preceding activities as well; e.g., the patient needs to be prepared* exactly 1 day before *the* `treatment` *takes place. This represents a* ***Time Lag between two Activities*** *(TP1) between the start of* `prepare patient` *and that of* `perform treatment`*. Hence,* `prepare patient` *needs to be scheduled in accordance with the* appointment *of the* `treatment`*. Additionally, the* `preparation` *of the patient may only be performed during opening hours of the anesthetics department, i.e., from* Monday till Friday between 8am and 4pm*. In turn, this represents a* ***Schedule Restricted Element*** *(TP5). It further restricts possible execution times of* `prepare patient` *and hence those of the* `treatment` *itself. Next, due to a high utilization of the treatment room, the* `preparation` *of the treatment should* not take more than 1 hour*; otherwise, the* `treatment` *might be delayed. This represents a maximum* ***Duration*** *(TP2) for activity* `prepare treatment`*. In turn, the* `treatment` *itself may take* between 1 and 4 hours*, which represents a minimum as well as a maximum* ***Duration*** *(TP2), i.e., an interval. Finally, during the execution of activity* `perform aftercare`*, different drugs are given to the patient, according to a* treatment plan *defined by activity* `perform treatment`*. Such a treatment plan may state, for example, that drug A must be administered* every day at 8 am, 1 pm, and 6 pm*, and drug B* every two hours, except when drug A is administered within the same hour*. This plan represents a* ***Periodicity*** *(TP10); more precisely, it constitutes the underlying periodicity rule of the* ***Periodicity*** *represented by activity* `perform aftercare`*.*

Example 3 demonstrates the diversity and complexity of the temporal perspective of business processes. It further emphasizes the need for precise and formal semantics of the different time patterns. For example, consider the *Fixed Date Element* of activity `perform treatment` in combination with the *Time Lag* between activities `prepare patient` and `perform treatment` and the *Schedule Restricted Element* of activity `prepare patient`. When combining these three temporal constraints, one may deduce that the `treatment` itself may only be performed from Tuesday to Saturday. In turn, this restriction needs to be considered when arranging the appointment for the `treatment` during activity `make appointment`. As a prerequisite for determining such implicit restrictions, a *precise formal semantics* needs to be defined for each time pattern. Only then, algorithms for automatically detecting such complex interdependencies can be developed. As another example, assume that in certain cases it becomes necessary to reschedule the appointment of activity `perform treatment` when `informing the patient` (e.g., due to an overbooking of the treatment room). For this case, a write access of activity `inform patient` to data object `date` must be added, which allows this activity to modify the appointment for the *Fixed Date Element* of activity `perform treatment`. However, when changing this appointment, it is not fully clear how to treat activity `prepare patient`. In particular, the latter must be executed exactly one day prior to activity `perform treatment`. As a result, if `inform patient` is executed after activity `prepare patient`, it must be decided whether or not the process instance may be continued as is. Particularly, it must be determined whether the `treatment` shall be rescheduled

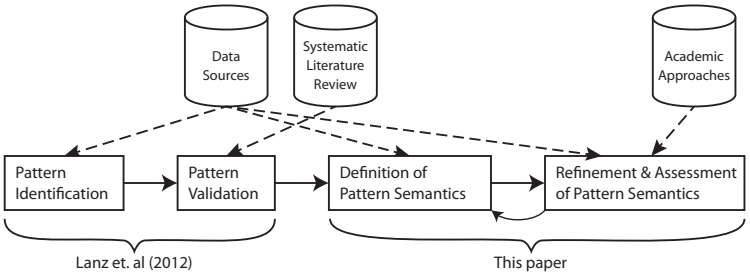

Figure 5: Overall research method

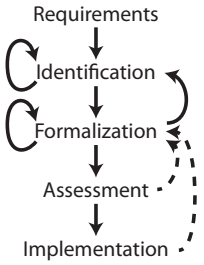

Figure 6: Applied research method

within the same day (i.e., the *Time Lag* between `prepare patient` and `perform treatment` is still satisfied) or the `preparation` of the patient must be redone in accordance with the new `appointment` for the `treatment`. Again, such questions can be only be addressed by the PAIS based on precise and formal semantics of the time patterns.

As illustrated by Example 3, there exist numerous variants of the time patterns. To cope with this variability and to keep the number of patterns manageable, *design choices* (DC) allow for their parametrization [17, 18]. For example, whether a time lag represents a minimum value, maximum value or both (i.e., an interval) constitutes a design choice of the *Time Lags between Activities* pattern (TP1). We denote a specific combination of design choices of a time pattern as a *pattern variant*. Usually, PAISs only support a subset of the design choices (i.e., variants) of a particular pattern. However, note that each pattern variant has a slightly different semantics. For example, to be able to exactly define the semantics of pattern *Duration* (TP2), we must specify whether the duration corresponds to a minimum value, a maximum value or both, i.e., the semantics of TP2 depends on the actual pattern variant.

Finally, when applying a specific pattern variant to a process schema, the restriction represented by the pattern occurrences needs to be determined. In particular, a pattern variant only represents the type of restriction, whereas the restriction itself is configured when applying the pattern to a process schema. When applying pattern *Duration* (TP2) to an activity, for example, the duration restriction needs to be specified, e.g., "`maximum duration: 1 hour`" for activity `prepare treatment` in Example 3. Likewise, it needs to be defined how the particular `appointment` required in the context of pattern *Fixed Date Element* for activity `perform treatment` shall be determined at run-time (cf. Example 3). For this purpose, *parameter values* allow determining the specific temporal constraint represented by a pattern occurrence. Respective parameter values may either be set at build-time (i.e., all process instances share the same value) or at run-time when creating or executing process instances. In the latter case, the parameter value is stored in a data object of the process instance, which will be written either when creating the process instance or executing one of its activities (e.g., data object `date` in Fig. 4). The stored data value is then evaluated by the corresponding pattern instance at run-time. For this purpose, a pattern occurrence may be linked to a data object. In Fig. 4, for example, the *Fixed Date Element* (cf. activity `perform treatment`) is linked to data object `date`.

3. Research Method

This section describes the research method we applied to develop the formal semantics for the time patterns. This includes the initial identification of the time patterns [18] as well as the definition of their semantics (cf. Fig. 5). We further describe essential requirements for defining a pattern semantics and present the procedure applied for their elicitation.

3.1. Initial Pattern Identification & Validation

Selected time patterns were first introduced in [17]. In turn, [18] provided an informal description of all time patterns as well as empirical evidence for them. Selection criteria were "patterns covering temporal aspects relevant for the modeling and control of business processes and activities, respectively" [18]. Other

Healthcare Domain (Women's Hospital) [25, 26]	88 processes
Healthcare Domain (Internal Medicine) [27, 28, 29]	93 processes
Automotive Domain [30, 31]	55 processes
On-demand Air Service [32, 33]	2 processes
Airline Catering Services [34]	1 process
Transportation [35]	9 processes
Software Engineering [36]	10 processes
Event Marketing[a]	1 process
Financial Service[a]	10 processes
Municipality Processes [37]	4 processes
Student Administration[a]	73 processes
Software Asset Management[a] [38]	56 processes
Build-to-Order Machine Manufacturer[a]	37 processes
Total:	439 processes

[a] Due to copyright & secrecy restrictions some of these processes cannot be made publicly available.

Table 2: Data sources [18]

process aspects (e.g., data flow) or time support features (e.g., escalation management, verification) were not considered and are thus outside the scope of this paper.

To create a solid basis for the time patterns, first of all, a candidate pattern list was created based on the knowledge and experiences of the authors in the field. Furthermore, *design choices* were introduced for parameterizing the patterns to keep their number manageable. Then, large sets of industrial cases, process schemas, and process documentation were analyzed (cf. Table 2) to provide empirical evidence for each time pattern and to refine the pattern list. Finally, this resulted in a set of 10 time patterns [18]. To validate them and to assess their relevance and completeness, additionally, we conducted a systematic literature review [18]. Finally, existing approaches and tools were evaluated regarding the support of the time patterns.

3.2. Pattern Semantics: Requirements, Procedure & Verification

This section describes essential requirements for defining formal time pattern semantics, the procedure we applied for identifying pattern semantics, and the overall evaluation process chosen. In this context, we follow a design science approach as proposed by Hevner et al. [23, 24] which is outlined in Fig. 6.

Requirements. To be useful for engineers as well as researchers, formal pattern semantics should be as generic as possible, while at the same time being as specific as necessary to avoid ambiguities (R1). In particular, if formal pattern semantics are too restrictive, they will not be applicable in many practically relevant cases. Conversely, if it is too permissive, it may allow for too many ambiguities and hence be of limited benefit (especially when trying to detect inconsistencies). Furthermore, formal pattern semantics must consider all *pattern variants* that may be configured based on different combinations of design choices (R2). To ensure that different implementations of a particular pattern share the same underlying semantics, its formal semantics needs to precisely define the effects, the respective pattern shall have on process enactment, as well as its interactions with different elements of the control flow perspective (R3). For the same reason, the impact, process instance data has on pattern semantics, needs to be specified (R4). Finally, as emphasized, any pattern semantics will only be useful for PAIS engineers if it is independent of a particular process modeling language (e.g., BPMN vs. EPC) or process modeling paradigm (e.g., imperative vs. declarative) (R5).

Identification. To identify the semantics of each time pattern, we first analyze the data sources used for pattern identification as well as three additional sources (cf. Table 2). Note that most data sources were gathered by us over the last decade. Moreover, in most cases we were involved in the elicitation of the processes (e.g., interviews with domain experts). In all these cases, we paid attention to thoroughly and completely record respective processes. We not only described the control flow, data flow, and organizational perspectives, but also documented additional requirements and business rules like exceptional cases, exception

handling rules, compliance rules, and temporal constraints along with the processes. Moreover, we ensured that these additional requirements are accurately described including their intended purpose and effects as declared by domain experts. This enables us to use the data sources to analyze the intended semantics of each pattern occurrence. Next, for each time pattern, we combine the collected semantics of all its occurrences–considering the respective pattern variant–to derive the overall semantics of the pattern. Finally, if appropriate, we remove unnecessary restrictions from the overall pattern semantics.

Formalization. Workflow patterns have been defined based on languages with an inherent formal semantics (e.g., pi-calculus [39]). However, such formalisms cannot be used to define the semantics of time patterns without significant extensions since they do not consider the temporal perspective. Beyond that, different techniques, commonly used for describing the temporal properties of a system, exist. Examples include temporal logics [40], timed Petri nets [41], timed automata [42], and execution traces [21]. Basically, all these techniques can be used for describing the formal semantics of time patterns.

On one hand, the definitions of the formal semantics should be independent of a particular process modeling language or paradigm. On the other, they should be closely related to the execution semantics of processes. To cope with this trade-off, this paper uses *execution traces* as basis for formalizing pattern semantics. An *execution trace* represents a 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. Sect. 4.1). Based on this, we can specify which execution traces (i.e., process instances) are *valid* according to the time patterns found in the respective process schema. Thus, we obtain an approach that is independent of the particular process modeling language and paradigm; yet, we are still able to precisely define pattern semantics. By associating the defined pattern semantics with a process meta model, the effects a particular pattern has on process enactment, as well as its interactions with different elements of the meta model, can be derived in a precise and formal way.

Taking the definition of the formal semantics we reconsider each pattern occurrence observed in any data source (cf. Table 2), and compare the described semantics with the formal definitions; i.e., we revisit the *Identification* step (cf. Fig. 6). If the semantics of one of the pattern occurrences in the data sources is not precise enough, we discuss it with the co-authors not involved in the identification of this particular occurrence. Whenever necessary, we discuss a pattern occurrence with the respective stakeholder involved in the initial elicitation of the data source to ensure that the pattern semantics meets the one the domain experts had in mind. If the formal semantics of a particular time pattern needs to be modified, we implement the required change (i.e., we repeat steps *Identification* and *Formalization*). We repeat these steps until all occurrences of every pattern observed in any data source are covered by the formal pattern semantics. This ensures that the formal semantics indeed cover all pattern occurrences in the data sources.

Implementation and Testing. In a final step, we use the formal semantics of the time patterns as foundation for implementing selected time pattern variants in the ATAPIS Toolset [43]. With this toolset we aim to provide a comprehensive framework enabling the specification, enactment and monitoring of processes with temporal constraints in PAISs. Moreover, we are currently expanding this implementation such that it may serve as a reference implementation for the time patterns. We further use it to once more test that the formal descriptions accurately describe the semantics of the time patterns. For the latter purpose, we model several processes from the data sources using the ATAPIS Toolset. For each of the resulting process schemas, we then use ATAPIS to simulate the execution of multiple process instances. Finally, for each pattern occurrence and for each of the simulated process instances we compare the resulting execution log with the expected results and the formal semantics of the time patterns. This shows that both the formal semantics of the time patterns and their implementation work as intended.

4. On the Formal Semantics of Time Patterns

Sect. 4.1 first defines basic notions (e.g., *event* and *trace*) used throughout this paper. Sect. 4.2 then provides the formal semantics of each time pattern.

S_2:

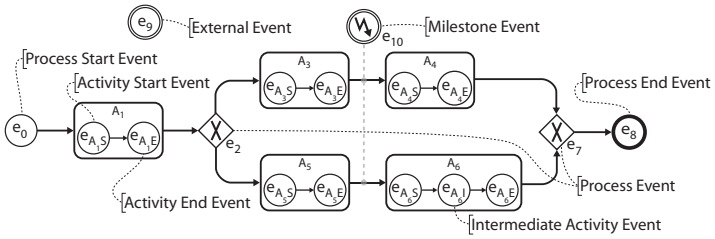

Figure 7: Events occurring during a process

4.1. Temporal Execution Traces

We use the notion of *event* as a general term for something that happens during the execution of a process instance, e.g., the start and end of an activity, executed in the context of a particular process instance, constitute such events (cf. Fig. 7). External events (e.g., reaching a milestone or receiving a message) may exist as well. Finally, intermediate events may be triggered inside a long-running activity (cf. Fig. 7).

Definition 1 (Event, Event occurrence). *Let $\mathcal{PS}$ be the set of all process schemas and $\mathcal{E}$ be the set of all events. Then: The set of all events that may occur during the execution of an instance I of process schema $S \in \mathcal{PS}$ is denoted as $\mathcal{E}_S \subseteq \mathcal{E}$. (Note that, without loss of generality, we assume unique labeling of events.) Let further $\mathcal{C}$ be the total set of absolute time points. Then:*

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

denotes the occurrence of event $e \in \mathcal{E}$ at time point $t \in \mathcal{C}$, i.e., t defines the exact point in time at which event e occurred. 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. Finally, 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 $\varphi = (e, t)$. ◊

Based on events and event occurrences, we define the notion of *temporal execution trace* (*trace* for short) which is inspired by [21]. We assume that all events related to the execution of a process instance are recorded in a temporal execution trace together with a time stamp designating their time of occurrence.Formally:

Definition 2 (Temporal Execution Trace). *Let $\mathcal{PS}$ be the set of all process schemas. Then: For schema $S \in \mathcal{PS}$, $\mathcal{Q}_S$ denotes the set of all temporal execution traces producible on S. Let Φ_S be the set of all event occurrences that may occur during the execution of S and $\varphi = (e, t) \in \Phi_S$ denote the occurrence of event e at time point t. A temporal execution trace $\sigma_S \in \mathcal{Q}_S$ is defined as ordered set of event occurrences φ_i:*

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

Without loss of generality, we assume that events in σ_S do not occur at exactly the same point in time. ◊

Note that we consider events to be instantaneous. Thus—with sufficient clock precision—one may assume that there are no events of a trace occurring at exactly the same time. We make this assumption to foster comprehensibility of the pattern semantics and to reduce ambiguities. However, it does not constitute a restriction of the presented semantics, and may be dropped without having to change pattern semantics. In the latter case, however, it might not always be clear which event exactly violates a particular pattern occurrence (i.e., there might be multiple equivalent reasons for the violation of a pattern occurrence).

Since the pattern semantics are defined based on events and event occurrences, we define nodes, activities, and gateways based on events. Remember that nodes may represent both activities and gateways (cf. Sect. 2.1). Respective definitions are based on the events that may occur during the execution of the particular node. Since gateways (e.g., XOR-splits, AND-joins) merely serve structuring purposes, without loss of generality, we may assume that the processing of a gateway does not take any time during process

execution. If a gateway consumes time during its execution (e.g., evaluating an XOR decision), this may simply be represented by an activity directly preceding the gateway. Note that, this is no particular restriction of the presented pattern semantics, but only serves the purpose of simplifying some of the formulas.

Definition 3 (Node, Activity, and Gateway). *Let $\mathcal{PS}$ be the set of all process schemas and $\mathcal{N}_S \subset 2^{\mathcal{E}_S}$ be the set of all nodes process schema $S \in \mathcal{PS}$ refers to.*

A node *$n \in \mathcal{N}_S$ is defined as unique, non-empty set of events $n \subset \mathcal{E}_S$ (i.e., $\forall n \in \mathcal{N}_S : n \neq \emptyset$). Thereby, the sets of events of all nodes belonging to S are disjoint: $\forall n, m \in \mathcal{N}_S, n \neq m : n \cap m = \emptyset$. An occurrence of a node n in a temporal execution trace σ_S is marked by the occurrence of at least one of the respective events, i.e., $\exists e \in n, \exists t \in \mathcal{C} : \varphi_e = (e, t) \in \sigma_S$. In turn, an* activity *is a special kind of node with a single start and single end event (cf. Sect. 2.1). Let $\mathcal{A}_S \subseteq \mathcal{N}_S$ be the total set of activities S refers to. An activity $a \in \mathcal{A}_S$ contains (at least) a unique start event e_{a_S} and a unique end event e_{a_E}, i.e.,*

$$\forall a \in \mathcal{A}_S : \exists e_{a_S}, e_{a_E} \in \mathcal{E}_S : \{e_{a_S}, e_{a_E}\} \subseteq a$$

An occurrence of an activity a is marked by the occurrence of at least both the start event e_{a_S} and the end event e_{a_E} in trace σ_S, i.e., $\exists t_1, t_2 \in \mathcal{C} : \varphi_{a_S} = (e_{a_S}, t_1) \in \sigma_S \wedge \varphi_{a_E} = (e_{a_E}, t_2) \in \sigma_S$.

Finally, a gateway *is a special kind of node represented by single event e denoting the time the gateway is processed by the system, i.e., for a gateway $c \in \mathcal{N}_S$ it holds $|c| = 1$.* ◊

Example 4 illustrates these definitions.

Example 4 (Temporal Execution Trace). *Consider Fig. 7. An example of a temporal execution trace on process schema S_2 may be as follows:*[2]

$$\begin{aligned}\sigma = \langle &(e_0, 2), (e_{A_1S}, 5), (e_{A_1E}, 10), (e_2, 11), (e_{A_5S}, 15), (e_{A_5E}, 18),\\ &(e_{10}, 19), (e_{A_6S}, 30), (e_{A_6I}, 32), (e_{A_6E}, 37), (e_7, 38), (e_8, 39)\rangle\end{aligned}$$

This trace indicates that the corresponding process instance was started (i.e., e_0) at time 2. At time 5, activity A_1 was started (e_{A_1S}). In turn, A_1 was completed at time 10 (e_{A_1E}); i.e., A_1 had duration $10 - 5 = 5$. Next, event e_2 occurred at time 11 and the lower path was chosen, which is indicated by the occurrence of start event e_{A_5S} related to activity A_5. Remember that none of the events of the upper path occurs within the trace as the respective path had been skipped. Further note that the occurrence of XOR-split e_2 is indicated by a single event, i.e., processing of the gateway does not take any time. After completing activity A_5 (e_{A_5E}), milestone event e_{10} was triggered. Next, A_6 was started at time 30. Furthermore, during the execution of A_6, the intermediate activity event e_{A_6I} was triggered. Finally, after completion of A_6, events e_7 and e_8 were triggered completing the process instance.

When defining the semantics of the time patterns, for some of them the iteration of a loop structure needs to be taken into account.

Example 5 (Cyclic Elements). *Consider process schema S_3 in the upper part of Fig. 8. It comprises two nested loops. Thereby, the inner loop L_2 contains two alternative branches either executing activity A_6 or A_7. As indicated in Fig. 8, the time lag between A_6 and A_7 must be a Cyclic Element (TP9) since A_6 and A_7 cannot be executed within the same iteration of the respective loop. In turn, time pattern TP9 requires the target activity to be contained in an iteration succeeding the one to which the source activity belongs to. Assume that the cyclic element only restricts the maximum time lag between directly succeeding iterations (Design Choice K[a]), i.e., the respective maximum time lag must be obeyed solely if A_7 is executed in an iteration of Loop L_2 directly after A_6 without executing another instance of A_6, A_7 or A_{10} in between. To formalize this semantics, it is not sufficient to introduce an iteration counter for activities. Regarding Fig. 8 it cannot always be detected whether another iteration of L_1 has been executed between the two instances of A_6 and A_7. Hence, the current iteration of the outer loop L_1 must be considered as well.*

[2] In the following, without loss of generality, we use integers starting at 0 for representing absolute time points $c \in \mathcal{C}$.

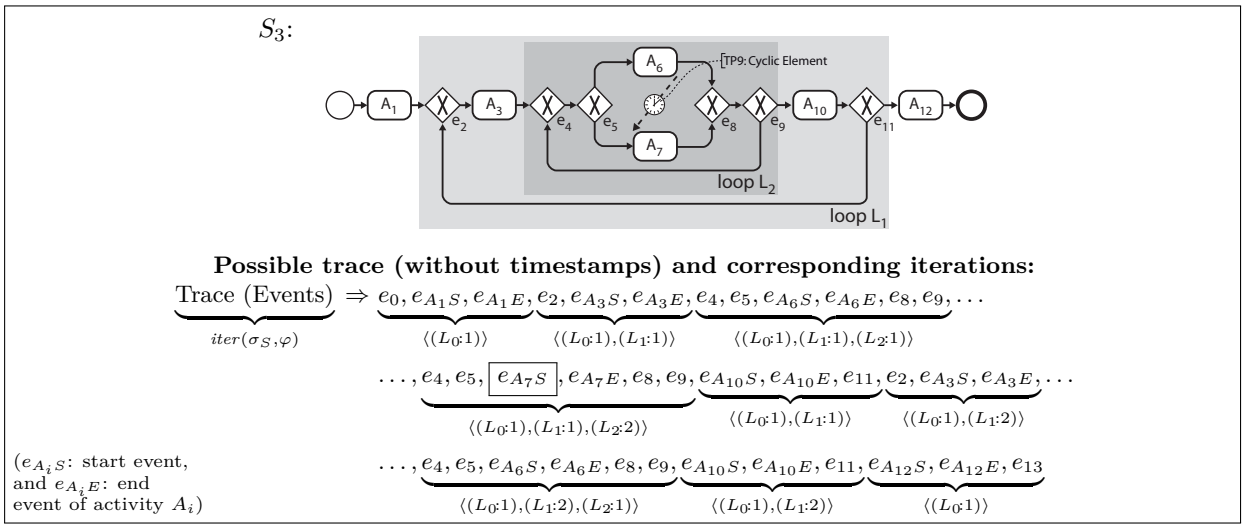

Figure 8: Nested loops and iterations

Generally, one must always consider the iteration of the inner-most loop in respect to the iteration of any surrounding loop. Regarding process schema S_3 in Fig. 8, for example, an instance of activity A_6 may belong to the 3rd iteration of loop L_2 within the 2nd iteration of loop L_1.

For the sake of simplicity and to be able to uniquely identify the iteration of each activity, we presume well-nested loops (see Fig. 8). If a process schema is not well-nested, in most practically relevant cases, we can transform it into a behaviour-equivalent, well-nested representation [44, 45, 46]. Nevertheless, we are aware that this might not be possible for all non-well-nested schemas [44]. Consequently, the notion of (loop) iteration is defined as follows:

Definition 4 (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 $iter(\sigma_S, \varphi)$ returns the current iteration of the inner-most loop surrounding event φ^e. Formally:

$$iter : \mathcal{Q}_S \times \Phi_S \mapsto \mathcal{I}_S \quad with \quad iter(\sigma_S, \varphi) = \langle (L_0 : n_{L_0}), \ldots, (L_k : n_{L_k}) \rangle$$

and $L_1, \ldots, L_k$ being the loop structures surrounding event φ^e. ◊

Example 6 (Iteration). *Consider Fig. 8: Below process schema S_3, a temporal execution trace is shown. Moreover, for each event in the trace, the value of $iter(\sigma_S, \varphi)$ is given; e.g., regarding the occurrence of event e_{A_7S} during the 2nd iteration of loop L_2 within the 1st iteration of loop L_1, we obtain*

$$iter\,(\sigma_S, (e_{A_7S}, \cdot)) = \langle (L_0 : 1), (L_1 : 1), (L_2 : 2) \rangle$$

This is indicated in the second line of the execution trace depicted in Fig. 8.

Based on Def. 4, Table 3 summarizes useful notions that will facilitate the following formalization of the time patterns. Function $succ(I)$ determines the next iteration after iteration I. In turn, the adjacency

Let $\sigma_S = \langle \varphi_1, \ldots, \varphi_n \rangle \in \mathcal{Q}_S$ be a temporal execution trace on process schema S. Then:

Iteration I occurs in trace σ_S, denoted as $\sigma_S \vdash I$, iff $\exists \varphi \in \sigma_S : iter(\sigma_S, \varphi) = I$

For iteration $I = \langle (L_0 : n_{L_0}), \ldots, (L_k : n_{L_k}) \rangle$, its next iteration $succ(I)$ is defined as

$$succ(I) \equiv \big\langle (L_0 : n_{L_0}), \ldots, (L_{k-1} : n_{L_{k-1}}), (L_k : n_{L_k} + 1) \big\rangle$$

Iteration I_a is adjacent to iteration I_b, denoted as $I_a \parallel I_b$, iff

- I_b represents the first iteration of an inner loop of I_a, i.e., for $n \geq 1$:

$$I_a = \langle (L_0 : n_{L_0}), \ldots, (L_k : n_{L_k}) \rangle \wedge I_b = \langle (L_0 : n_{L_0}), \ldots, (L_k : n_{L_k}), (L_{k+1} : 1), \ldots, (L_{k+n} : 1) \rangle$$

- or I_a represents the last iteration of an inner loop of I_b, i.e., for $n \geq 1$:

$$I_a = \big\langle (L_0 : n_{L_0}), \ldots, (L_{k-n} : n_{L_{k-n}}), \ldots, (L_k : n_{L_k}) \big\rangle \wedge I_b = \big\langle (L_0 : n_{L_0}), \ldots, (L_{k-n} : n_{L_{k-n}}) \big\rangle \wedge$$
$$\forall_{m \in [k-n,k]} : n_{L_m} = \max \big\{ x | \sigma_S \vdash \big\langle (L_0 : n_{L_0}), \ldots, (L_{m-1} : n_{L_{m-1}}), (L_m : x) \big\rangle \big\}$$

Iteration I_b is a succeeding iteration of iteration I_a, denoted as $I_a < I_b$, iff

$$(succ(I_a) = I_b) \vee \big(\exists I' \in \mathcal{I}_S : \big(succ(I') = I_b \vee I' \parallel I_b \big) \wedge I_a < I' \big)$$

Table 3: Useful notions

operator $\parallel$ allows determining adjacent iterations–two iterations are adjacent if one of them is the first or last iteration of a nested loop with respect to the other iteration. Finally, the $<$-operator determines whether the second iteration is (transitively) succeeding the first one.

Since a particular event may occur multiple times within a trace (e.g., at the presence loops), we further define function $occur(\sigma_S, e)$. It returns all event occurrences of an event e within a trace:

Definition 5 (Trace Occurrences). *Let $S \in \mathcal{PS}$ be a process schema and σ_S be a trace on S. Then: $occur(\sigma_S, e)$ corresponds to a function returning all occurrences $\varphi = (e, \cdot)$ of event $e \in \mathcal{E}_S$ within σ_S. Formally:*

$$occur : \mathcal{Q}_S \times \mathcal{E}_S \mapsto 2^{\Phi_S} \quad with \quad occur(\sigma_S, e) = \{\varphi | \exists t \in \mathcal{C} : \varphi = (e, t) \in \sigma_S\}$$

◊

Def. 5 implies that $occur(\sigma_S, e) = \emptyset$ holds, if $\nexists t \in \mathcal{C} : (e, t) \in \sigma_S$, e.g., in case the respective path has been skipped (cf. Example 4).

Similar to events, activities may occur multiple times within a trace. In turn, each instance of an activity is marked by the occurrence of its start and end event (cf. Def. 3). For this purpose, we introduce function $occur(\sigma_S, a)$, which returns all instances of a particular activity a (i.e., occurrences of the start and end events) within a trace:

Definition 6 (Activity Occurrence). *Let $S \in \mathcal{PS}$ be a process schema and σ_S be a trace on S. Then: $occur(\sigma_S, a)$ returns all occurrences (φ_S, φ_E) of the start and end event $e_{a_S}, e_{a_E} \in \mathcal{E}_S$ of activity $a \in \mathcal{A}_S$ (i.e., $\{e_{a_S}, e_{a_E}\} \subseteq a$) within trace σ_S. Thereby, two occurrences of events e_{a_S} and e_{a_E} belong to the same activity instance if they occur within the same iteration. Formally:*

$$occur : \mathcal{Q}_S \times \mathcal{A}_S \mapsto 2^{\Phi_S \times \Phi_S}$$

with $occur(\sigma_S, a) = \{(\varphi_S, \varphi_E) \in \Phi_S \times \Phi_S \mid$
$$\varphi_S \in occur(\sigma_S, e_{a_S}) \wedge \varphi_E \in occur(\sigma_S, e_{a_E}) \wedge iter(\sigma_S, \varphi_S) = iter(\sigma_S, \varphi_E)\}$$
Note that the start event e_{a_S} of an activity instance always occurs before the corresponding end event e_{a_E}, i.e., $\forall (\varphi_S, \varphi_E) \in occur(\sigma_S, a) : \varphi_S^t < \varphi_E^t$ ◊

For the sake of readability, we will omit subscript S in the formulas if the respective process schema becomes evident from the given context.

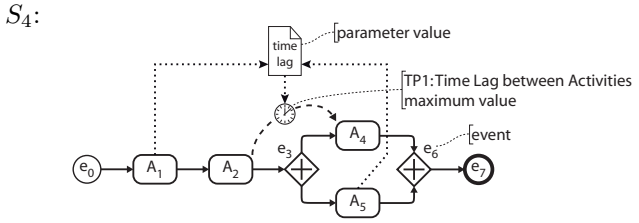

Figure 9: Run-time parameter value and concurrent change

4.2. Time Pattern Semantics

The semantics of the time patterns can now be defined by characterizing the temporal execution traces σ_S that may be produced by instances of a process schema S. In particular, we specify which temporal execution traces satisfy the temporal constraints expressed by any time pattern applied to S. Formally:

Definition 7 (Temporal Compliance). *A temporal execution trace $\sigma_S \in \mathcal{Q}_S$ is* temporally compliant *with the set of temporal constraints defined on process schema $S \in \mathcal{PS}$ if and only if σ_S is temporally compliant with each of the temporal constraints corresponding to S. In this context, temporal compliance of a trace with a particular temporal constraint on S is defined by the semantics of the respective time pattern.* $\diamond$

Based on Defs 1-7, we are able to describe the formal semantics of all time patterns. Thereby, we do not consider at which point in time the parameter values of a pattern occurrence are set (i.e., at build-time, process creation-time, or run-time; *Design Choice A*) as this does not affect formal pattern semantics. However, we consider which parameter value shall be valid in the context of a particular pattern instance; i.e., if the parameter value of a pattern occurrence may be modified during run-time (e.g., deadlines may be changed), we specify which of the values shall be relevant for a particular instance of a pattern occurrence when defining pattern semantics.

Example 7 (Run-time Parameter Values). *Consider process schema S_4 in Fig. 9: S_4 contains a maximum time lag between activities A_2 and A_4. The value of the time lag is provided by data object* `time lag` *at run-time. In turn, this data object is written twice during process execution by activities A_1 and A_5. Note that A_5 may be executed (or—more important—be completed) either before, during, or after executing A_4. Depending on the completion time of A_5, the time lag may use a different version of the parameter value, i.e., a different version of the value of data object* `time lag`*. Therefore, it is important to precisely state which of these versions shall be valid for a particular instance of the time lag.*

To structure our formalization effects, we bundle time patterns with similar or related foundations as they have same or similar formalization.

The first set of time patterns with similar foundations comprises patterns TP1 (*Time Lags between two Activities*), TP2 (*Duration*), TP3 (*Time Lags between Arbitrary Events*), and TP9 (*Cyclic Elements*). These patterns restrict the relative time distance between pairs of events that may occur during the execution of a process instance. The kind of relative time distance expressed constitutes a pattern-specific design choice (*Design Choice D*); i.e., the time distance may be a minimum value, a maximum value, or an interval. The kind of events to which a specific pattern can be applied may be restricted by the pattern itself; e.g., a time lag between activities may only be applied to the start or end events of two activities. Furthermore, only pairs of event occurrences, whose iterations are consistent with the pattern, may be considered; e.g., for an activity *Duration* (TP2; *Design Choice C[a]*), the occurrences of the start and end event of the respective activity must belong to the same iteration.

Definition 8 (Relative time distance between events). *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 events related to the time distance, i.e., the events for which the time distance is specified. Temporal compliance of a given trace σ with a relative time distance between events e_X and e_Y is then defined as follows:*

All valid pairs of event occurrences $\varphi = (e_X, \cdot)$ and $\psi = (e_Y, \cdot)$ must satisfy the relative time distance as it is effective at the occurrence time of target event ψ. Thereby, a pair of event occurrences is valid if their iterations are consistent with the respective pattern. This is expressed by pattern-specific function $\text{valid} : \mathcal{Q} \times \Phi \times \Phi \rightarrow \textit{Boolean}$. *In turn, function* $\text{distance} : \mathcal{Q} \times \mathcal{E} \times \mathcal{E} \times \mathcal{C} \rightarrow \left[\mathcal{D}\right]$[3] *represents the parameter value of the respective pattern occurrence between events e_X and e_Y as it is effective at time t (cf. Example 7). Finally, whether or not a pair of event occurrences φ and ψ satisfies the relative time distance depends on the kind of temporal distance represented by the pattern (i.e., minimum distance (DC D[a]), maximum distance (DC D[b]), or time interval (DC D[c]); cf. Design Choice D). In turn, this is defined by function*

$$\text{compareR} : \mathcal{C} \times \mathcal{C} \times \left[\mathcal{D}\right] \rightarrow \textit{Boolean}$$

with[4]

$$\text{compareR}(\varphi^t, \psi^t, d) \equiv \begin{cases} \varphi^t + \min(d) \leq \psi^t & \textit{if minimum distance (DC D[a])} \\ \psi^t \leq \varphi^t + \max(d) & \textit{if maximum distance (DC D[b])} \\ \varphi^t + \min(d) \leq \psi^t \leq \varphi^t + \max(d) & \textit{if time interval (DC D[c])} \end{cases}$$

All patterns of the first pattern set (i.e., patterns TP1, TP2, TP3, and TP9) can now be formalized using the following Formula (1) and applying different restrictions to e_X, e_Y, and $\text{valid}(\sigma, \varphi, \psi)$*:*

$$\forall \varphi \in \textit{occur}(\sigma, e_X), \forall \psi \in \textit{occur}(\sigma, e_Y) : \text{valid}(\sigma, \varphi, \psi) \Rightarrow \text{compareR}(\varphi^t, \psi^t, \text{distance}(\sigma, e_X, e_Y, \psi^t)) \quad (1)$$

$\Diamond$

According to Formula (1), for each valid pair of occurrences of the source and target events, the time distance between them must be compared with the current parameter value of the respective pattern (according to the kind of time distance represented by the pattern; cf. *Design Choice D*). Note that only the parameter value (i.e., the version of the data value), which is valid at the time the target event occurred, is considered (cf. Example 7). Based on this general definition, the semantics of time patterns TP1, TP2, TP3, and TP9 can be defined as described in the following.

Time pattern TP1 (*Time Lags between Activities*) allows restricting the time span between the starting/ending instants of two activities A and B [18]; i.e., it allows expressing a minimum or maximum time distance between respective start/end events. Particularly, time lags may be used to specify minimum delays and maximum waiting times between succeeding activities. Moreover, time lags may not only be defined between directly succeeding activities, but between any two activities that may be conjointly executed in the context of a particular process instance, i.e., the activities must not belong to exclusive branches. As particular variants of this pattern, it may be specified whether a time lag represents a Start-Start relation (i.e., referring to the start events of two different activities), a Start-End relation, an End-Start relation, or an End-End relation (*Design Choice E*).

Note that the definition of the pattern semantics needs to be valid in connection with loops as well. As illustrated by Fig. 10, considering time lags between two activities (TP1) of which one resides inside a loop and the other one outside this loop (e.g., the time lag between A_1 and A_3 in Fig. 10) is a challenging task due to the unclear semantics of such a scenario. This becomes even more complicated when considering nested loops. As example consider the time lag between A_1 and A_6 in Fig. 10 for which—on the face of it—various semantics exist. *First*, a time lag whose source activity is contained in one loop and whose target activity is inside a nested loop (cf. Fig. 10) might only apply to the very first or very last iteration of the nested loop. However, this effectively makes time lags of little or no use in the context of multiple nested loops. When considering the time lag between A_3 and A_6 in Fig. 10, it would only apply to the first-ever iteration of A_6 when using the above semantics. Particularly, if the loop containing A_3 is repeated the time lag no longer applies as A_6 has already been executed. *Second*, the time lag may be applied to all iterations

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

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

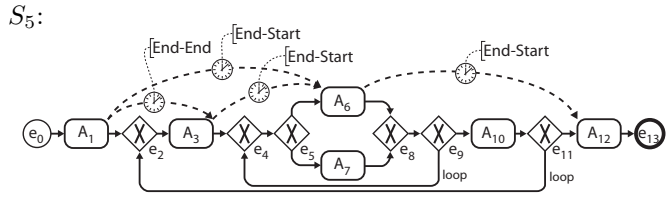

Figure 10: Time lags and loops

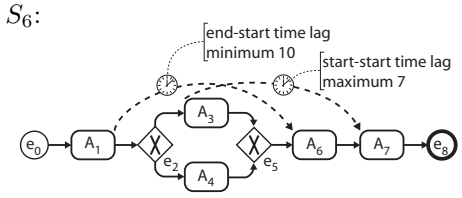

Figure 11: Illustration of Pattern Semantics 1

of the inner loop. This case, however, results in different semantics for minimum and maximum time lags. Particularly, a minimum time lag would only apply to the first iteration of the target activity as it is trivially fulfilled for all other iterations of the latter. In turn, a maximum time lag would effectively only apply to the last iteration of the target activity as all other iterations automatically fulfill the time lag. *Third*, such an "inbound" time lag might apply to the first iteration of the inner loop with respect to any iteration of an outer loop. *Fourth*, the time lag may be augmented with meta information describing the iterations it applies to. Note that the latter represents a generalization of the third option. However, we did not find any evidence for this kind of time lag semantics in the data sources. Moreover, all occurrences of pattern *Time Lags between Activities* found in the data sources indicating the necessity for one of the first two options, can be easily expressed using the *third* option. Finally, note that the same holds for time lags whose source activity lies inside a nested loop and whose target activity is outside that loop (cf. Fig. 10).

As a consequence, the definition of the TP1 pattern semantics restricts the application of a time lag to adjacent iterations of respective loops, i.e., it restricts an "inbound" time lag to the first iteration and an "outbound" time lag to the last iteration of a loop. According to our experience this constitutes the most intuitive semantics. If necessary, this semantics can be extended to consider specific iterations of the respective loops as well (i.e., option *four*). However, as we have not found any evidence for this kind of semantics this would make the respective pattern semantics unnecessarily complex.

Pattern Semantics 1 (TP1: Time Lags between Activities). *The semantics of time pattern TP1 can be defined based on Formula (1) by applying the following restrictions to* e_X, e_Y, *and* $\text{valid}(\sigma, \varphi, \psi)$*:*

(a) Depending on the kind of relation a time lag represents (i.e., Start-Start, Start-End, End-Start, or End-End; cf. Design Choice E), event e_X *either corresponds to the start or end event of the source activity. Likewise,* e_Y *corresponds to the start or end event of the time lag's target activity.*

(b) The two activities—or more precisely their events—must either belong to the same or an adjacent iteration, i.e., the first or last iteration of an inner loop. Consequently, it holds:

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

For example, restriction (a) expresses that for an End-Start relation between two activities, e_X corresponds to the end event of the source and e_Y to the start event of the target activity; i.e., the constraint restricts the time lag between the end event of the source and the start event of the target activity.

Example 8 (Pattern Semantics 1). *We illustrate Pattern Semantics 1 for process schema* S_6 *from Fig. 11.* S_6 *contains a minimum time lag of 10 between the end of* A_1 *and the start of* A_6*, and a maximum time lag of 7 between the start of* A_3 *and the start of* A_7*. Based on* S_6*, for example, traces* σ_1*-*σ_3 *can be produced.*

$$\begin{aligned}\sigma_1 = \langle &(e_0, 0), (e_{A_1S}, 1), (e_{A_1E}, 3), (e_2, 4), (e_{A_3S}, 10), (e_{A_3E}, 12),\\ &(e_5, 13), (e_{A_6S}, 14), (e_{A_6E}, 15), (e_{A_7S}, 17), (e_{A_7E}, 19), (e_8, 20)\rangle\end{aligned}$$

$$\begin{aligned}\sigma_2 = \langle &(e_0, 0), (e_{A_1S}, 2), (e_{A_1E}, 5), (e_2, 6), (e_{A_4S}, 8), (e_{A_4E}, 10),\\ &(e_5, 11), (e_{A_6S}, 13), (e_{A_6E}, 19), (e_{A_7S}, 25), (e_{A_7E}, 27), (e_8, 28)\rangle\end{aligned}$$

$$\sigma_3 = \langle (e_0, 0), (e_{A_1S}, 4), (e_{A_1E}, 7), (e_2, 8), (e_{A_3S}, 10), (e_{A_3E}, 18), \\ (e_5, 19), (e_{A_6S}, 20), (e_{A_6E}, 22), (e_{A_7S}, 23), (e_{A_7E}, 29), (e_8, 30)\rangle$$

Trace σ_1 is temporally compliant (cf. Def. 7) with the set of temporal constraints defined on S_6 (cf. Fig. 11). In particular, the time lag between A_1 and A_6 is satisfied as

$$\text{compareR}(3, 14, \text{distance}(\sigma, e_{A_1S}, e_{A_6E}, 14)) \equiv \\ \varphi^t_{e_{A_1E}} + \min(\text{distance}(\sigma, e_{A_1S}, e_{A_6E}, 14)) = 3 + 10 = 13 \leq \varphi^t_{e_{A_6S}} = 14$$

holds. Further, the time lag between A_3 and A_7 is satisfied:

$$\varphi^t_{e_{A_7S}} = 17 \leq \varphi^t_{e_{A_3S}} + \max(\text{distance}(\sigma, e_{A_3S}, e_{A_7S}, 17)) = 10 + 7 = 17.$$

In turn, trace σ_2 is not temporally compliant with the temporal constraints on S_6. Though the time lag between A_3 and A_7 is trivially fulfilled, as A_3 is not executed, the one between A_1 and A_6 is not satisfied:

$$\varphi^t_{e_{A_1E}} + \min(\text{distance}(\sigma, e_{A_1S}, e_{A_6E}, 15)) = 5 + 10 = 15 \not\leq \varphi^t_{e_{A_6S}} = 13.$$

Finally, trace σ_3 is not temporally compliant with S_6. In this case, the time lag between A_1 and A_6 is satisfied. However, this does not apply to the time lag between A_3 and A_7, as it holds:

$$\varphi^t_{e_{A_7S}} = 23 \not\leq \varphi^t_{e_{A_3S}} + \max(\text{distance}(\sigma, e_{A_3S}, e_{A_7S}, 23)) = 10 + 7 = 17.$$

We now consider the semantics of time pattern TP3 (*Time Lags between Arbitrary Events*), which restricts the time lag between two arbitrary discrete events [18]. Note that certain events cannot be bound to the start or end of an activity or process; e.g., there may be events triggered by external sources not controllable by the PAIS (e.g., receiving a message from a partner process). In addition, there may be events not bound to a specific activity, e.g., event "delivery of all parts" requires several processes to be completed. Finally, there may be events triggered inside a long-running activity, e.g., when reaching a milestone during the execution of a long-running activity or sub-process.

Pattern Semantics 2 (TP3: Time Lags between Arbitrary Events). *The semantics of TP3 can be expressed based on Formula (1) by applying the following restrictions to e_X, e_Y, and* valid(σ, φ, ψ)*:*
(a) Both e_X and e_Y constitute arbitrary events.
(b) Like for Pattern Semantics 1, respective events must either belong to the same or to an adjacent iteration. Hence, it holds:

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

When considering Pattern Semantics 2, it can be observed that TP3 constitutes a generalization of TP1 [18]. In particular, restriction (a), as specified in the context of Pattern Semantics 1, is dropped.

Time pattern TP9 (*Cyclic Elements*) restricts the time lag between activities A and B across loop iterations.[5] As discussed in [18], TP9 constitutes a variation of TP1. However, instead of enforcing the two activity instances to be executed within the same or adjacent iterations of a loop, we require the instance of the target activity to be contained in an iteration succeeding the one the source activity belongs to. Similar to TP1, TP9 may describe a Start-Start, Start-End, End-Start, or End-End relation between the respective activities (*Design Choice E*). Finally, an instance of TP9 may not only restrict the time lag between two activity instances of directly succeeding loop iterations, but also between two subsequent activity instances belonging to arbitrary loop iterations (*Design Choice K*).

[5] Note that A and B may represent the same activity.

Pattern Semantics 3 (TP9: Cyclic Elements). *The semantics of time pattern TP9 can be expressed based on Formula (1) by applying the following restrictions to* e_X, e_Y, *and* $\text{valid}(\sigma, \varphi, \psi)$*:*

(a) For e_X *and* e_Y *the same restrictions as in the context of Pattern Semantics 1 apply.*

(b) According to Design Choice K the activities (and their corresponding events) of a cyclic element must belong to directly succeeding iterations (DC K[a]) or to arbitrary, but succeeding iterations (DC K[b]):

$$\text{valid}(\sigma, \varphi, \psi) \equiv \begin{cases} (\, succ(iter(\sigma, \varphi)) = iter(\sigma, \psi)) \vee \\ \quad (succ(iter(\sigma, \varphi)) \parallel iter(\sigma, \psi)) & \text{if the iterations shall directly succeed each other (DC K[a])} \\ \big(\, iter(\sigma, \varphi) < iter(\sigma, \psi)\big) \wedge & \text{if two subsequent activity instance belonging to arbitrary iterations (DC K[b])} \\ \quad \big(\nexists \omega \in occur(\sigma, \varphi^e) \cup occur(\sigma, \psi^e) : \\ \qquad iter(\sigma, \varphi) < iter(\sigma, \omega) < iter(\sigma, \psi)\big) & \end{cases}$$

Restriction (b) can be interpreted as follows: Depending on *Design Choice K*, the target event must either occur in the directly succeeding iteration of the source event or in one of its adjacent iterations (*Design Choice K[a]*) or in an arbitrary iteration succeeding the iteration of the source event (*Design Choice K[b]*). In the first case, the succeeding iteration of the source event (i.e., $succ(iter(\sigma, \varphi))$) is adjacent or equal to the iteration of the target event. In the second case, the target event must belong to an iteration succeeding the one of the first event, i.e., $iter(\sigma, \varphi) < iter(\sigma, \psi)$. However, there must not be any occurrence of either the source event φ^e or the target event ψ^e between these two iterations.

Time pattern TP2 (*Durations*) expresses that an activity, activity set, process, or set of process instances must obey a specific duration restriction [18]. Durations result from both waiting (e.g., activity is suspended) and processing times during activity execution. However, a duration does not cover the time span the activity is offered in user worklists, but has not been started yet (i.e., the time span between the activation and start of the activity; cf. Fig. 3). The semantics of TP2 is similar to Pattern Semantics 1-3 since it considers the relative time distance between events.

Pattern Semantics 4 (TP2: Durations). *For an activity or process (Design Choices C[a] and C[c]), respectively, the semantics of TP2 is defined based on Formula (1). In particular, for an activity the value of function* distance *is given by function* $\text{duration} : \mathcal{Q} \times \mathcal{A} \times \mathcal{C} \rightarrow \big[\mathcal{D}\big]$*, which represent the actual parameter value of the pattern occurrence (i.e.,* $\text{distance}(\sigma_S, \varphi_S, \varphi_E, t) \equiv \text{duration}(\sigma_S, a, t)$*;* $(\varphi_S, \varphi_E) \in occur(\sigma_S, a)$*). In turn, for a process the value of function* distance *is provided by function* $\text{duration} : \mathcal{Q} \times \mathcal{PS} \times \mathcal{C} \rightarrow \big[\mathcal{D}\big]$*. Furthermore, the following restrictions apply to* e_X, e_Y *and* $\text{valid}(\sigma, \varphi, \psi)$*:*

(a) Depending on whether the pattern is applied to an activity or process, e_X *(*e_Y*) corresponds to the start (end) event of the activity or process, respectively.*

(b) Both events of a valid pair must belong to the same activity (process) instance and hence to the same iteration (cf. Def. 6), i.e.,

$$\text{valid}(\sigma, \varphi, \psi) \equiv (iter(\sigma, \varphi) = iter(\sigma, \psi))$$

In turn, for an activity set $A \subset \mathcal{A}$ *(Design Choice C[b]), Formula (1) needs to be extended to consider event sets as well. Let* E_X *represent the set of start events and* E_Y *the set of end events of all activities from A. Furthermore, function* $\text{duration} : \mathcal{Q} \times 2^{\mathcal{A}} \times \mathcal{C} \rightarrow \big[\mathcal{D}\big]$ *represents the parameter values of the pattern occurrence applied to activity set A as it is effective at time t. Then:*

- *For a minimum duration (Design Choice D[a]), the following must hold: For each valid pair of occurrences of events* $e_X \in E_X$ *and* $e_Y \in E_Y$*, there exists at least one pair of event occurrences within the same iteration that satisfies the respective minimum duration:*

$$\begin{aligned} &\forall a \in A : \forall (\mu_S, \mu_E) \in occur(\tau, a) \Rightarrow \\ &\quad [\, \exists e_X \in E_X, \exists e_Y \in E_Y : \exists \varphi \in occur(\sigma, e_X), \exists \psi \in occur(\sigma, e_Y) : \\ &\qquad (iter(\sigma, \mu_S) = iter(\sigma, \varphi) \wedge \text{valid}(\sigma, \varphi, \psi)) \Rightarrow \varphi^t + \min(\text{duration}(\sigma, A, \psi^t)) \leq \psi^t \,] \end{aligned} \tag{2}$$

Each iteration may have several valid event pairs. However, only one of them must obey the given minimum duration. Note that the duration of the activity set corresponds to the maximum time distance between any events of the set.

- *A maximum duration (Design Choice D[b]) must be met by all valid pairs of event occurrences $e_X \in E_X$ and $e_Y \in E_Y$:*

$$\forall e_X \in E_X, \forall e_Y \in E_Y : \forall \varphi \in occur(\sigma, e_X), \forall \psi \in occur(\sigma, e_Y) : \\ \text{valid}(\sigma, \varphi, \psi) \Rightarrow \psi^t \leq \varphi^t + \max(\text{duration}(\sigma, A, \psi^t)) \tag{3}$$

- *For a time interval (Design Choice D[c]), Formulas (2) and (3) must both be satisfied.*

For handling a set of process instances (i.e., cross-process constraints; Design Choice C[d]), the above formulas must be extended to consider different process instances. In this case, E_X (E_Y) represents the start (end) events of all process instances of the given set. We omit respective formulas here.

Regarding Pattern Semantics 1-4, it is noteworthy that none of the described formalizations requires that all referred events are present in a temporal execution trace; e.g., a maximum time lag between two activities is trivially fulfilled if the target activity has never been executed in the context of the process instance. This is of particular relevance at the presence of alternative branches, where not all events of a process schema may actually occur during the execution of a process instance (cf. Example 4). As example consider process schema S_5 (cf. Fig. 10). If A_6 is never executed for a particular process instance (i.e., the lower path is always selected) the time lag between A_3 and A_6 does not need to be obeyed (i.e., it is trivially fulfilled for this process instance). If it is required that both activities connected by a time lag must always be conjointly executed by any process instance, this must be ensured by other means (e.g., control flow).

As another set of time patterns with similar foundations we consider TP4 (*Fixed Date Element*) and TP7 (*Validity Period*). Both patterns refer to absolute points in time and may be applied to either an activity or a process (*Design Choice C*). Furthermore, they allow restricting the earliest start, latest start, earliest completion, or latest completion date of the activity or process (*Design Choice F*).

Definition 9 (Absolute time point of an event). *Let $\mathcal{C}$ be the total set of absolute time points. Compliance of a given trace σ with an absolute time point for event e is defined as follows: All occurrences $\varphi = (e, \cdot)$ of the event must satisfy the absolute time point as it is effective at the time the event occurred. Thereby, function* $\text{date} : \mathcal{Q} \times \mathcal{E} \times \mathcal{C} \rightarrow \mathcal{C}$ *represents the parameter value of the pattern occurrence being effective at time t. Finally, whether or not an event occurrence φ satisfies an absolute time point depends on the kind of date (i.e., earliest/latest start date, earliest/latest completion date) restricted by the respective constraint (Design Choice F). This is expressed by function*

$$\text{compareA} : \mathcal{C} \times \mathcal{C} \rightarrow \textit{Boolean}$$

with

$$\text{compareA}(\varphi^t, t) \equiv \begin{cases} t \leq \varphi^t & \textit{if the earliest start or completion date is restricted (DC F[a] or F[c])} \\ \varphi^t \leq t & \textit{if the latest start or completion date is restricted (DC F[b] or F[d])} \end{cases}$$

TP4 and TP7 can then be formalized using Formula (4) by applying different functions for $\text{date}(\sigma, e, t)$.

$$\forall \varphi \in occur(\sigma, e) : \text{compareA}(\varphi^t, \text{date}(\sigma, e, \varphi^t)) \tag{4}$$

Depending on the kind of process element (i.e., activity or process; Design Choice C) and the kind of date restricted by the constraint (i.e., earliest/latest start, earliest/latest completion; Design Choice F), event e either corresponds to the start or end event of the activity (process). ◊

Time pattern TP4 (*Fixed Date Element*) allows expressing that a particular activity (process) instance must be executed in relation to a particular date (e.g., a deadline) [18]; i.e., for a particular activity or

S_7:

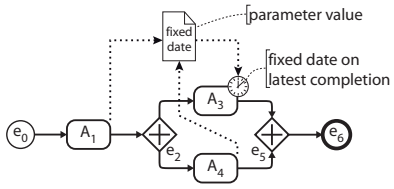

Figure 12: Illustration of Pattern Semantics 5

S_8:

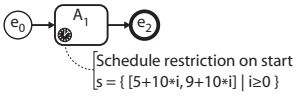

Figure 13: Illustration of Pattern Semantics 7

process, it can be specified whether it must to be started after, started before, completed after, or completed that date (*Design Choice F*). Moreover, note that the parameter value of a fixed date element is specific to a process instance, i.e., it is not known before creating the process instance.

Pattern Semantics 5 (TP4: Fixed Date Element). *The semantics of time pattern TP4 can be expressed with Formula (4). Thereby,* $\mathrm{date}(\sigma, e, \varphi^t)$ *corresponds to a function returning the parameter value of the pattern occurrence being effective for process instance* σ *at the time of event occurrence* $\varphi = (e, t)$.

Example 9 (Pattern Semantics 5). *We illustrate Pattern Semantics 5 for process schema* S_7 *(cf. Fig. 12).* S_7 *contains a Fixed Date Element constraining the latest completion date of activity* A_3*. In turn, the parameter value of the Fixed Date Element (i.e., the value of function* date*) is determined by activity* A_1 *and may be modified by activity* A_4*. For example, the following two traces may be produced based on* S_7*.*

$$\sigma_4 = \Big\langle (e_0,0),(e_{A_1S},2),(e_{A_1E},4)_{\{\mathrm{date}(\sigma,e_{A_3E},t\geq 4)\leftarrow 8\}},(e_2,5),(e_{A_3S},6),$$
$$(e_{A_4S},7),(e_{A_3E},8),(e_{A_4E},10)_{\{\mathrm{date}(\sigma,e_{A_3E},t\geq 10)\leftarrow 12\}},(e_5,11),(e_6,12)\Big\rangle$$
$$\sigma_5 = \Big\langle (e_0,0),(e_{A_1S},2),(e_{A_1E},4)_{\{\mathrm{date}(\sigma,e_{A_3E},t\geq 4)\leftarrow 12\}},(e_2,5)(e_{A_4S},6),$$
$$(e_{A_4E},7)_{\{\mathrm{date}(\sigma,e_{A_3E},t\geq 7)\leftarrow 8\}},(e_{A_3S},8),(e_{A_3E},9),(e_5,10),(e_6,12)\Big\rangle$$

In this context, $(e_{A_1E},4)_{\{\mathrm{date}(\sigma,e_{A_3E},t\geq 4)\leftarrow 8\}}$ *indicates that after the occurrence of event* e_{A_1E} *the value of* $\mathrm{date}(\sigma, e_{A_3E}, \cdot)$ *is changed to* 8*, i.e.,* $\forall t \geq 4 : \mathrm{date}(\sigma, e_{A_3E}, t) = 8$ *(cf. trace* σ_4*).*

σ_4 *complies with the Fixed Date Element constraining* A_3 *since the corresponding parameter value is set to* 8 *by* A_1 *and is not modified by* A_4 *before completing* A_3*. Hence, for the completion event of* A_3 *it holds*

$$\varphi^t_{A_3E} = 8 \quad \leq \quad \mathrm{date}(\sigma, e_{A_3E}, \varphi^t_{A_3E}) = 8.$$

In turn, σ_5 *is not temporally compliant with the Fixed Date Element constraining* A_3*. Initially, the parameter value of this Fixed Date Element is set to* 12 *by* A_1*. However, before completing* A_3*,* A_4 *is executed, changing the parameter value of the Fixed Date Element to* 8*. Hence, for the completion of* A_3 *it holds*

$$\varphi^t_{A_3E} = 9 \quad \not\leq \quad \mathrm{date}(\sigma, e_{A_3E}, \varphi^t_{A_3E}) = 8.$$

Time pattern TP7 (*Validity Period*) allows restricting the life time of an activity (process) to a particular *validity period* [18]. This becomes relevant, for example, in the context of process schema evolution [47] to restrict the remaining life time of an obsolete process schema as well as to schedule the rollout of the new schema version. Since TP7 may prohibit the execution of an activity or process (schema), its parameter value needs to be known prior to process instance execution and cannot be modified during run-time.

Pattern Semantics 6 (TP7: Validity Period). *The semantics of TP7 can be expressed based on Formula (4). Thereby, the value of* $\mathrm{date}(\sigma, e, t)$ *is independent of both the current process instance* σ *and time* t*, i.e.,* $\mathrm{date}(\cdot, e, \cdot) \equiv const$.

Note that Pattern Semantics 5 and 6 are similar. As only difference, for Pattern Semantics 6 the value of function $\text{date}(\cdot, e, \cdot)$ is independent of the particular process instance and the current point in time. In turn, for Pattern Semantics 5 the actual return value of $\text{date}(\sigma, e, t)$ may be different for each process instance and time point. However, the effects of patterns TP4 and TP7 on process execution are different. For example, TP7 must be verified prior to process instance creation, whereas TP4 can only be evaluated during run-time. Consequently, the semantics of time patterns TP4 and TP7 (i.e., Pattern Semantics 5 and 6) are different.

In the following we consider the semantics of time pattern TP5 (*Schedule Restricted Elements*) [18]. TP5 allows restricting the possible execution times of an activity (or process) through a *schedule*. In turn, a schedule constitutes an abstract representation of a possibly infinite set of *time slots* described in terms of a finite expression (e.g., Mo-Fr, 8:00-17:00). Since we do not want to restrict the representation of such a schedule (e.g., [48, 49]), we require that it can be materialized as a set of intervals on the absolute time points $\mathcal{C}$ (i.e., as a set of time slots).

Definition 10 (Schedule). *A schedule s is a possibly infinite set of continuous intervals (i.e., time slots) on the absolute time points $\mathcal{C}$. Formally:*

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

$\Diamond$

Each time slot of a schedule specifies a time frame during which the respective event of the activity (process) may occur. Particularly, a *Schedule Restricted Element* always restricts both the earliest and the latest start or the earliest and the latest completion date of the activity (process) (*Design Choice F*).

Based on Def. 10, the semantics of time pattern TP5 (*Schedule Restricted Element*) can be defined:

Pattern Semantics 7 (TP5: Schedule Restricted Elements). *Trace σ is temporally compliant with a schedule s for event e of an activity if for all event occurrences $\varphi = (e, \cdot)$ the respective time stamp φ^t is contained within one of the time slots of the schedule. Thereby, function* $\text{schedule} : \mathcal{Q} \times \mathcal{E} \times \mathcal{C} \rightarrow \left[\mathcal{C}\right]$ *represents the parameter value (i.e., the schedule) of the pattern occurrence being effective at time t. Formally:*

$$\forall \varphi \in \mathit{occur}(\sigma, e) : \exists [t_{min}, t_{max}] \in \text{schedule}(\sigma, e, \varphi^t) : t_{min} \leq \varphi^t \leq t_{max} \tag{5}$$

Depending on the kind of process element (Design Choice C) and the kind of date restricted by the constraint (Design Choice F), event e either corresponds to the start or end event of the activity (process).

Example 10 (Pattern Semantics 7). *Fig. 13 illustrates Pattern Semantics 7 along process schema S_8. S_8 represents a process with a schedule restricting the start of A_1; the schedule is given by expression $s = \{[5 + 10 * i, 9 + 10 * i] | i \geq 0\} = \{[5, 9], [15, 19], [25, 29], \ldots\}$. Examples of traces on S_8 are σ_6 and σ_7:*

$$\sigma_6 = \langle (e_0, 0), (e_{A_1 S}, 2), (e_{A_1 E}, 9), (e_2, 10) \rangle$$
$$\sigma_7 = \langle (e_0, 0), (e_{A_1 S}, 6), (e_{A_1 E}, 11), (e_2, 12) \rangle$$

σ_6 is not temporally compliant with the respective schedule restriction on A_1 as

$$\nexists [t_{min}, t_{max}] \in s = \{[5, 9], [15, 19], [25, 29], \ldots\} : t_{min} \leq \varphi^t_{A_1 S} = 2 \leq t_{max}.$$

In turn, σ_7 is temporally compliant with the schedule restriction since

$$\exists [t_{min}, t_{max}] \in s = \{[5, 9], [15, 19], [25, 29], \ldots\} : t_{min} \leq \varphi^t_{A_1 S} = 6 \leq t_{max}.$$

In particular, $[t_{min}, t_{max}] = [5, 9] \in s$.

Design Choice U
U) Execution time frame of the activity (process) and the given time frame may be compared in different ways. (a) Overlapping time frames (b) Execution time frame contained in given time frame (c) Start of execution within given time frame (d) End of execution within given time frame

Figure 14: Design Choice U for time pattern TP6

Time-based Restrictions (TP6) allow restricting the number of executions of an activity, a set of activities, a process, or a set of process instances (*Design Choice G*) within a given time frame [18], e.g., to express that a particular activity must not be executed more than once per day. TP6 can be formalized by comparing the number of executions of the respective activities (processes) within the given time frame on one hand with a given minimum or maximum number of executions (*Design Choice H*) on the other. However, when considering the semantics of TP6, a subtle difference between the various pattern variants can be observed, which has not been covered by the original design choices yet. To be able to completely define the semantics of TP6, *Design Choice U* (cf. Fig. 14) allows specifying how the execution time frames of the respective activities (processes) and the reference time frame shall be compared. Based on *Design Choice U*, the number of executions of a particular activity within a given time frame can be defined.

Definition 11 (Executions per time frame). *Let σ_S be a trace on process schema $S \in \mathcal{PS}$. Then: Function compareT$([t_{min}, t_{max}], [\varphi_S^t, \varphi_E^t])$ expresses whether the execution time frame $[\varphi_S^t, \varphi_E^t]$ of an activity occurrence (φ_S, φ_E) and the given time frame $[t_{min}, t_{max}]$ fulfil the restriction set out by Design Choice U. Formally:*

$$\text{compareT} : \left[\mathcal{C}\right] \times \left[\mathcal{C}\right] \rightarrow \mathit{Boolean}$$

with $\text{compareT}([t_{min}, t_{max}], [\varphi_S^t, \varphi_E^t])$

$$\equiv \begin{cases} \left([t_{min}, t_{max}] \cap [\varphi_S^t, \varphi_E^t] \neq \emptyset\right) & \textit{the two time frames are overlapping (DC U[a])} \\ \left([\varphi_S^t, \varphi_E^t] \subseteq [t_{min}, t_{max}]\right) & \textit{execution time frame contained in } [t_{min}, t_{max}] \textit{ (DC U[b])} \\ \left(\varphi_S^t \in [t_{min}, t_{max}]\right) & \textit{start of execution within } [t_{min}, t_{max}] \textit{ (DC U[c])} \\ \left(\varphi_E^t \in [t_{min}, t_{max}]\right) & \textit{end of execution within } [t_{min}, t_{max}] \textit{ (DC U[d])} \end{cases}$$

Furthermore, function executions returns the number of occurrences of an activity $a \in \mathcal{A}_S$ within a given time frame $[t_{min}, t_{max}]$. Formally:

$$\mathit{executions} : \mathcal{Q}_S \times \mathcal{A}_S \times \left[\mathcal{C}\right] \mapsto \mathbb{N}_0$$

with $\mathit{executions}(\sigma_S, a, [t_{min}, t_{max}]) \equiv |\{(\varphi_S, \varphi_E) \in \mathit{occur}(\sigma_S, a) \mid \mathit{compareT}([t_{min}, t_{max}], [\varphi_S^t, \varphi_E^t]) = \mathit{true}\}|$

◊

Example 11 (Executions per time frame). *Consider process schema S_9 (cf. Fig. 15). Trace σ_8, which is illustrated in the right part of Fig. 15, can be produced based S_9.*

$$\begin{aligned} \sigma_8 = \langle & \ldots, (e_{A_3S}, 3), \ldots, (e_{A_6S}, 5), (e_{A_6E}, 9), \ldots, (e_{A_6S}, 14), (e_{A_6E}, 18), \ldots, (e_{A_6S}, 23), \\ & (e_{A_3E}, 24), \ldots, (e_{A_3S}, 27), (e_{A_6E}, 28), \ldots, (e_{A_6S}, 31), (e_{A_6E}, 35), \ldots, (e_{A_6S}, 38), \\ & (e_{A_6E}, 42), \ldots, (e_{A_6S}, 45), (e_{A_3E}, 47), \ldots, (e_{A_6E}, 49), \ldots \rangle \end{aligned}$$

Regarding the number of times A_6 is executed during the first and second iteration of A_3, we obtain the

following values of function executions$(\sigma_S, a, [t_{min}, t_{max}])$*:*

$$executions(\sigma_8, A_6, [3, 24]) = \begin{cases} 3 & \textit{the two time frames are overlapping (DC U[a])} \\ 2 & \textit{execution time frame contained in given time frame (DC U[b])} \\ 3 & \textit{start of execution within given time frame (DC U[c])} \\ 2 & \textit{end of execution within given time frame (DC U[d])} \end{cases}$$

$$executions(\sigma_8, A_6, [27, 47]) = \begin{cases} 4 & \textit{the two time frames are overlapping (DC U[a])} \\ 2 & \textit{execution time frame contained in given time frame (DC U[b])} \\ 3 & \textit{start of execution within given time frame (DC U[c])} \\ 3 & \textit{end of execution within given time frame (DC U[d])} \end{cases}$$

Based on Def. 11, we can specify the semantics of time pattern TP6 (*Time-based Restrictions*):

Pattern Semantics 8 (TP6: Time-based Restrictions). *For the sake of simplicity, we assume that a Time-based Restriction (TP6) is applied to a set of activities $A \subseteq \mathcal{A}$ related to the same process instance (Design Choice G[a]). In this case, temporal compliance of a given trace σ with a Time-based Restriction on A is defined as follows:*

- *A temporal execution trace σ is temporally compliant with a maximum number n of concurrent executions (Design Choices H[b], I[a]) of activities $A \subseteq \mathcal{A}$, iff*

$$\forall a \in A : \forall (\varphi_S, \varphi_E) \in occur(\sigma, a) : \left(\sum_{a' \in A} executions(\sigma, a', [\varphi_S^t, \varphi_E^t]) \right) \leq n$$

  For a minimum number of concurrent executions (Design Choice H[a]), the same formula applies when replacing "$\leq n$" by "$\geq n$".

- *Consider a Time-based Restriction on the number of executions per time period (Design Choice I[b]). Respective time periods may be represented as a schedule s (cf. Def. 10) without any gaps between the elements of the schedule. Then: A temporal execution trace σ is temporally compliant with a maximum number n of executions (Design Choice H[a]) of activities $A \subseteq \mathcal{A}$ per time period (i.e., per element of schedule s), iff:*

$$\forall [t_{min}, t_{max}] \in s : \left(\sum_{a \in A} executions(\sigma, a, [t_{min}, t_{max}]) \right) \leq n$$

  For a minimum number of executions per time period (Design Choice H[b]), the same formula applies when replacing "$\leq n$" by "$\geq n$".

For a set of activities belonging to different process instances (Design Choice G[b]) or a set of process instances (Design Choice G[c]), respective sums must be calculated taking the respective set of temporal execution traces into account. For the sake of brevity, respective formulas are omitted here.

The first part of Pattern Semantics 8 calculates the execution numbers of activities from set A within execution time frame $[\varphi_S^t, \varphi_E^t]$ of any occurrence (φ_S, φ_E) related to an activity $a \in A$. For each activity occurrence, this number is then compared with the given maximum number of concurrent executions. In turn, the second part of Pattern Semantics 8 calculates the execution numbers of activities $a \in A$ for each time slot of schedule s. It then compares this number with the given maximum number of executions per time period. The same applies in respect to a minimum number of executions.

Example 12 (Pattern Semantics 8). *Reconsider process schema S_9 from Fig. 15 and trace σ_8 from Example 11. Then, σ_8 is not temporally compliant with the time-based restriction between A_3 and A_6. In*

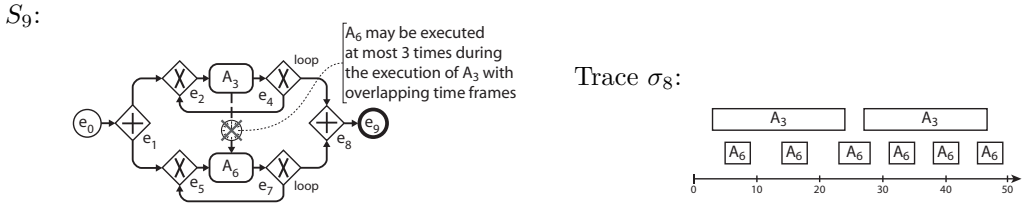

Figure 15: Illustration of Pattern Semantics 8

particular, for the first iteration (φ_S, φ_E) *of* A_3 *it holds (cf. Example 11):*

$$\sum_{a \in A} executions(\sigma_8, a, [\varphi_S^t, \varphi_E^t]) = executions(\sigma_8, A_3, [3, 24]) \overset{DC\ U[a]}{=} 3 \leq 3.$$

However, for the second iteration we obtain:

$$\sum_{a \in A} executions(\sigma_8, a, [\varphi_S^t, \varphi_E^t]) = executions(\sigma_8, A_3, [27, 47]) \overset{DC\ U[a]}{=} 4 \not\leq 3.$$

Time Pattern TP8 (*Time-dependent Variability*) allows varying the control flow depending on time aspects [18]. This may either be the execution time of a node (i.e., activity or gateway) or a time lag between two activities (e.g., if the first activity of a specific path is not started within a certain time frame, an alternative path will be chosen) (*Design Choice J*). Generally, TP8 restricts the set of events, a temporally compliant temporal execution trace may contain; i.e., events belonging to a selected path must occur within the trace, whereas events related to a deselected path must not occur within the respective trace (cf. Example 4). In particular, the semantics of a *time-dependent exclusive choice* (*Design Choice J[a]*) is the same as for the known workflow pattern *exclusive choice* [9]. Thereby, the decision about which path shall be taken is based on the execution time of the node representing the decision (e.g., XOR-split node). Similar considerations apply to a *time-dependent late binding*. In turn, the semantics of a *time-dependent deferred choice* (*Design Choice J[b]*) can be defined based on the *deferred choice* workflow pattern [9]. In detail, the semantics is the same as for a time lag between two activities (i.e., *Pattern Semantics 1*) plus the semantics of the *deferred choice* workflow pattern [9]. Since the semantics of TP8 can be defined based on the one of the respective workflow patterns (i.e., exclusive choice, late binding and deferred choice) no explicit semantics is provided for this pattern here.

Time Pattern TP10 (*Periodicity*) allows specifying periodically recurring sets of activities according to an explicitly defined periodicity rule [18]. Thereby, "periodically" implies some regularity, but does not necessarily mean equally distant. A periodicity rule describes the recurrence schema of the respective activities as well as some sort of exit conditions (e.g., every Monday and Wednesday at 11:30 until end of the year). It may be described using one or more schedules (cf. Example 16). Each of the activities of the periodicity is annotated with one of these schedules. Thereby, each activity has to be executed exactly once for each time slot of the respective schedule. Note that the schedules used to describe a periodicity rule may be not independent of each other, i.e., a particular schedule may define exceptions for another schedule. TP10 can be realized during run-time by using suitable combinations of time patterns TP1-6, TP8, and TP9. As example consider the periodicity depicted in Fig. 16 and its possible implementation shown inside the activity. Consequently, the semantics of pattern TP10 can be defined based on the one of the patterns used for realizing the respective periodicity during run-time. Therefore, no explicit semantics is provided for this pattern. However, note that even for simple periodicity rules, such a realization might lead to complex process schemas. Furthermore, periodicity rules are not always known at build-time, in the latter case it is not possible to pre-specify a corresponding process fragment. Therefore, Periodicity as additional layer of abstraction becomes necessary to describe respective processes in an understandable way, although semantics of TP10 may be specified based on the other time patterns.

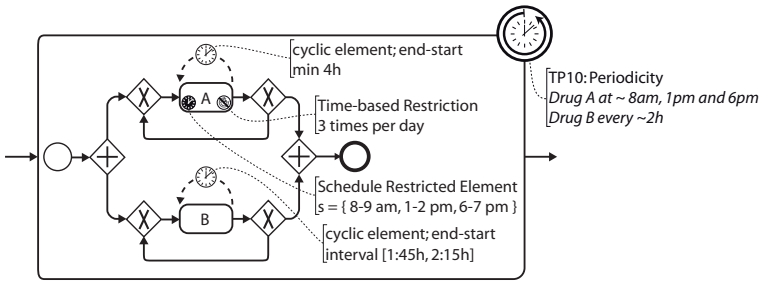

Figure 16: Illustration of Periodicity and possible realization

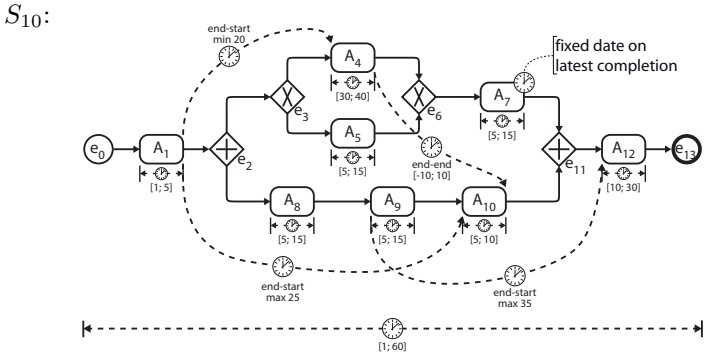

Figure 17: A process comprising different time patterns

4.3. Conclusion

Based on Pattern Semantics 1-8 we are able to check whether a given temporal execution trace is temporally compliant with all temporal constraints defined on the corresponding process schema. In this context, there is no particular restriction regarding the time patterns a process schema may contain, i.e., a single process schema may contain several occurrences of selected or all pattern variants. However, it must be ensured that for each possible path that may be selected during process execution, at least one execution trace being temporally compliant with all temporal constraints exists, i.e., the temporal constraints of all paths must be satisfiable. Otherwise, the temporal perspective of the process schema is inconsistent since 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. Formally:

Definition 12 (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 which may be executed during a single process instance) there exists at least one temporal execution trace $\sigma_S \in \mathcal{Q}_S$ being temporally compliant (cf. Def. 7) with the set of temporal constraints defined on S.* ◊

Example 13 (Temporal Consistency). *Consider process schema S_{10} as depicted in Fig. 17. Assume that for XOR-split e_3 the lower path is chosen. In this case, a possible execution trace is*

$$\begin{aligned}\sigma_{\{e_3 \rightarrow lower\}} = \langle &(e_0, 0), (e_{A_1 S}, 1), (e_{A_1 E}, 4), (e_2, 5), (e_3, 6), (e_{A_8 S}, 9), (e_{A_8 E}, 15), (e_{A_9 S}, 18),\\ &(e_{A_5 S}, 20), (e_{A_9 E}, 25), (e_{A_{10} S}, 28), (e_{A_5 E}, 35), (e_6, 36), (e_{A_7 S}, 37), (e_{A_{10} E}, 40),\\ &(e_{A_7 E}, 45), (e_{11}, 46), (e_{A_{12} S}, 48), (e_{A_{12} E}, 58), (e_{13}, 59)\rangle\end{aligned}$$

This particular execution trace complies with all temporal constraints defined on S_{10}. For example, the maximum process Duration is satisfied as $\varphi^t_{e_{13}} - \varphi^t_{e_0} = 59 \leq 60$ holds. Moreover, all duration constraints are satisfied. Finally, the two Time Lags between A_1 and A_{10} as well as between A_9 and A_{12}, which are applicable for this trace, are satisfied as $\varphi^t_{e_{A_1 E}} + 25 \geq \varphi^t_{e_{A_{10} S}}$ and $\varphi^t_{e_{A_9 S}} + 35 \geq \varphi^t_{e_{A_{12} S}}$ holds.

If the upper path is chosen for XOR-split e_3, no execution trace exists, which is temporally compliant with all temporal constraints on S_{10}. In particular, the path determining the shortest time to complete the process (i.e., the critical path) is given by $e_0, A_1, e_2, e_3, A_4, e_6, A_7, e_{11}, A_{12}, e_{13}$; this sums up to a minimum process duration of 72. In turn, this violates the maximum process Duration of 60.

Furthermore, the Time Lags between A_1 and A_4, A_1 and A_{10}, and A_4 and A_{10}, as well as the Duration constraints defined for A_4 and A_{10} cannot be satisfied at the same time: The earliest possible time at which A_4 may be completed after the completion of A_1 corresponds to $\varphi^t_{e_{A_1E}} + 20 + 30$. In turn, the latest possible time at which A_{10} may be completed after the completion of A_1 corresponds to $\varphi^t_{e_{A_1E}} + 25 + 10$. Accordingly, for the completion time of A_4 and A_{10} it holds $\varphi^t_{e_{A_{10}E}} - \varphi^t_{e_{A_1E}} \geq (\varphi^t_{e_{A_1E}} + 25 + 10) - (\varphi^t_{e_{A_1E}} + 20 + 30) = 35 - 50 = -15 \notin [-10, 10]$. Consequently, these constraints can never be satisfied at the same time, i.e., S_{10} is temporally inconsistent, as there exists an execution path for which no valid trace can be found.

Example 13 indicates the complexity of verifying the temporal consistency of a process schema in respect to its temporal constraints [19, 20]. Additional complexity is created by the fact that not all time patterns can be verified at build-time [18]. For example, the *Fixed Date Element* of an activity cannot be verified at build-time since not all parameter values are known before run-time [18]; i.e., the particular date is set during run-time. Moreover, some time patterns may have an infinite domain (e.g., *Schedule Restricted Elements*).

Overall, the presented formal time pattern semantics serve as a solid and sound framework for verifying the temporal consistency of a process schema with temporal constraints at build-time. Moreover, it may serve as basis for verifying and monitoring the temporal consistency of process instances during run-time.

5. Implementation

We are using the formal semantics of the time patterns as formal foundation for implementing selected time pattern variants in the ATAPIS Toolset[6] [43]. The ATAPIS Toolset is based on the AristaFlow BPM Suite—a process management system that exploits advanced process support features [1, 50]. We have chosen this technology since we also consider flexible run-time support of time-aware process instances (e.g., coping with dynamic changes as well [51, 52]). The ATAPIS Toolset allows specifying process schemas enriched with temporal constraints (cf. Sect. 5.1), which may then be checked for temporal consistency based on the defined pattern semantics (cf. Sect. 5.2). Moreover, it can be used to simulate the execution of a process instance, including the possibility to check it for constraint violations during run-time (cf. Sect. 5.3).

We have implemented support for the time patterns most commonly required in practice: *Time Lags between two activities* (TP1), *Durations* (TP2) of activities and processes (*Design Choices C[a, c]*), *Fixed Date Elements* (TP4), *Schedule Restricted Elements* (TP5), *Validity Periods* (TP7), *Time-Dependent Variability* (TP8) based on the execution time (*Design Choice J[a]*), and *Cyclic Elements* (TP9). Currently, we are investigating how periodicity rules may be specified to allow for more complex *Periodicities* (TP10).

5.1. Specifying Process Schemas with Temporal Constraints

Fig. 18 depicts the process editor of the ATAPIS Toolset displaying the process schema from Fig. 4. The ATAPIS process editor is based on the *AristaFlow Process Template Editor*, which we enhanced with capabilities and language elements required to capture and implement the selected time patterns [43]. Note that ATAPIS focuses on well-structured process schemas. This is no limitation of the time patterns or their semantics, but constitutes a restriction of AristaFlow's process modeling language [53].

In order to enable *Time Lags between two Activities* (TP1) and *Cyclic Elements* (TP9) we extended AristaFlow's process modeling language [53] with *time edges* (visualized through a dashed line as shown in Fig. 18). Considering Fig. 18, the dashed line between activities `prepare patient` and `perform treatment` with label $S[1d, 1d]S$ describes a time lag with a minimum and maximum time distance of 1 *day* between the start (S) of the two activities.

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

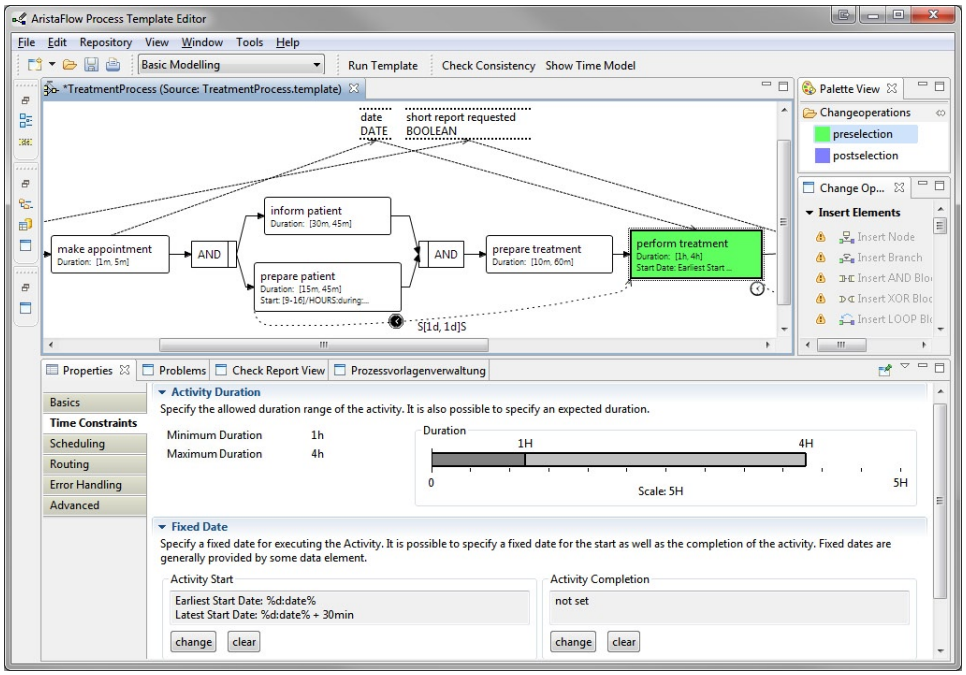

Figure 18: The ATAPIS Process Editor

In turn, time patterns restricting a particular activity (or the process itself) (i.e., *Duration*, *Fixed Date Element*, *Schedule Restricted Element*, and *Validity Period*) may be configured by using the *properties editor*. In Fig. 18 activity `prepare treatment` is currently selected and its properties are depicted in the lower part. In the upper section of the editor, the *Duration* (TP2) of the activity can be specified. In its lower part, two *Fixed Date Elements* (TP4) are specified for the activity. The one restricting the earliest start date of the activity retrieves its value from data element `date`. In turn, the one restricting the latest start date is set relatively to (i.e., 30 minutes after) the first one.

Similarly, a *Schedule Restricted Element* (TP5) may be specified for any activity. TP5 is implemented based on a language for representing collections of temporal intervals [54]; e.g., expression *[1-5]/days:during:weeks* represents a schedule comprising the first 5 days of each week (i.e., Monday–Friday).

To support *Time-Dependent Variability* (TP8) we provide a dedicated XOR decision rule for deciding which branch shall be taken based on the time the respective XOR-split is executed.

5.2. Checking Temporal Consistency at Build-Time

To ensure a robust and error-free execution of a process schema exhibiting temporal constraints, at build-time, the temporal consistency of the process schema (cf. Def. 12) should be checked. However, note that not all variants of the time patterns can be verified at build-time as discussed in Sect. 4.3.

To check whether a particular process schema is temporally consistent according to the pattern semantics (i.e., whether all its temporal constraints are satisfiable), ATAPIS maps it to a *Conditional Simple Temporal Network* (CSTN)—a problem known from artificial intelligence [55, 56, 51]. CSTN is an extension of Simple Temporal Networks [57] explicitly considering different execution paths. ATAPIS uses CSTN since it allows us to exploit and reuse provably sound *checking algorithms* for a well founded model representing temporal constraints. Moreover, CSTN allows capturing the complex interdependencies between constraints that cannot be properly captured in process schemas. Finally, CSTNs are similar to the execution traces used for defining patterns semantics and thus support the implementation of the provided formal pattern semantics.

Definition 13 (Conditional Simple Temporal Network). *A* Conditional Simple Temporal Network *(CSTN) is a 6-tuple* $\langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$*, where:*

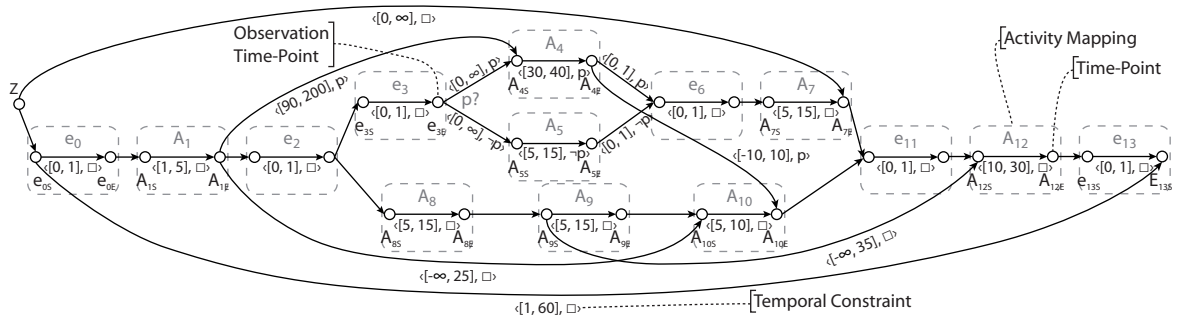

Figure 19: CSTN corresponding to the process schema from Fig. 17

- $\mathcal{T}$ *is a set of real-valued variables, called* time-points*;*
- P *is a finite set of* propositional letters *(or propositions);*
- $L : \mathcal{T} \rightarrow P^*$ *is a function assigning a* label *to each time-point in* $\mathcal{T}$*; a* label *is any (possibly empty) conjunction of (positive or negative) letters from* P*.*[7]
- $\mathcal{C}$ *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 time-points*;*
- $O : P \rightarrow \mathcal{OT}$ *is a bijection that associates an observation time-point to each propositional letter from* P*.*

Time-points 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 time-points X and Y must be at least x and at most y, i.e., $x \leq Y - X \leq y$. When reaching an *observation time-point*, a decision regarding possible execution paths is made. Formally speaking, when executing observation time-point P, the truth-value of the associated proposition (i.e., $O^{-1}(P)$) is determined. The *label* attached to each time-point (constraint) indicates possible executions of the CSTN, i.e., a particular time-point (constraint) will be only considered if its label is satisfiable in the respective instance. Fig. 19 depicts a CSTN represented as a graph whose nodes correspond to time-points and whose arcs correspond to constraints.

The solution to a CSTN is defined as follows [56]:

Definition 14 (Scenario and Solution). *Given a CSTN* $S = \langle \mathcal{T}, \mathcal{C}, L, \mathcal{OT}, \mathcal{O}, P \rangle$*, a* scenario *over set* P *is a function* $s_P : P \rightarrow \{true, false\}$ *that assigns a truth-value to each proposition in* P*.*

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

Based on this, consistency of a CSTN is defined as follows:

Definition 15 (Consistency of a CSTN). *A CSTN* S *is called* weakly consistent *iff for each scenario* s_P *at least one solution exists [55], i.e., a CSTN is weakly consistent iff all of its constraints are satisfiable.*

Basically, CSTN *time-points* are equivalent to events, whereas a time-point bound to a specific value is equal to an event occurrence (cf. Def. 1). Moreover, CSTN constraints are specified as inequalities of the form $Y - X \leq t$, where X and Y are time-points and t is a relative time distance. This is similar to the definition of a relative time distance between events (cf. Def. 8). Thus, it is possible to map a process schema with temporal constraints to a CSTN that preserves the semantics of the respective time patterns.

When mapping a process schema to a corresponding CSTN, first of all, the control flow of the process schema is mapped to the CSTN as illustrated in Fig. 20 [52, 51]. Note that each control flow element implicitly represents a temporal constraint, e.g., a control edge linking two activities is equivalent to a minimum time lag of 0 between the end of the source and the start of the target activity (cf. Fig. 20).

[7] In the following we use small Greek letters $\alpha, \beta, \ldots$ to denote arbitrary labels. The empty label is denoted by $\boxdot$.

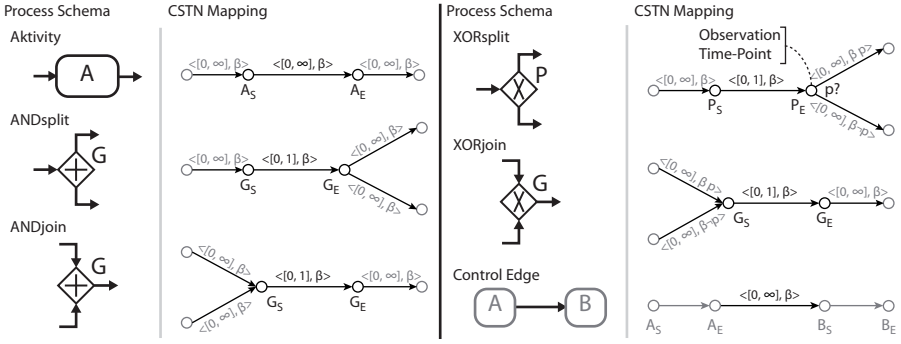

Figure 20: Process modeling elements and their mapping to CSTN

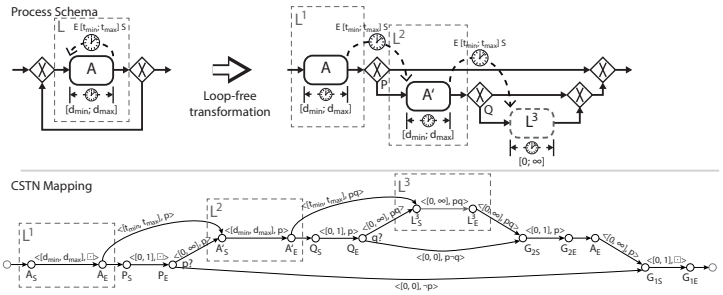

Figure 21: Loops and their mapping to CSTN

In a CSTN, each activity a_i is represented as a pair of time-points A_{iS} and A_{iE}, which correspond to the start and end event of the respective activity (cf. Fig. 20). Further, a constraint $\langle [0, \infty]_{A_{iS}A_{iE}}, \boxdot \rangle$ is added between A_{iS} and A_{iE} representing the execution of the respective activity, i.e., the constraint represents the fact that the end event of the activity always occurs after its start event. Similar to an activity, a gateway (i.e., AND-split, AND-join, XOR-split, and XOR-join) is represented by two time-points. This reflects the fact that—in reality—everything takes time. Note that this mapping difference, compared to Def. 3, does not influence the semantics of the time patterns. Furthermore, for an XOR-split, the end time-point (i.e., N_{iE}) is represented by an observation time-point P_{iE} with propositional letter p_i (cf. Fig. 20). As soon as the XOR-split is executed, it is decided which branch shall be taken. In the CSTN, this is reflected by observation time-point P_{iE} or, more precisely, the value of the associated propositional letter $p_i = O^{-1}(P_{iE})$, i.e., p_i = true reflects the decision for the `true`-branch and p_i = false the decision for the `false`-branch. Afterwards, the labels of all constraints and time-points corresponding to activities, gateways and control edges in the two XOR-branches are augmented by either p or $\neg p$ depending on the branch they belong to. For example, the label of a CSTN constraint $\langle [0, \infty]_{N_{iE}N_{jS}}, \beta \rangle$ belonging to the `true`-branch is set to $\gamma = \beta p$ and the one of the `false`-branch to $\gamma = \beta \neg p$ (cf. Fig. 20).

As example of this mapping re-consider Fig. 19. It depicts the CSTN of the process schema from Fig. 17. For the sake of readability, all edges without annotation are assumed to have bounds $\langle [0, \infty], \boxdot \rangle$.

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 [52] by replacing each loop structure by a block containing the respective number of clones of the original loop body. These clones are then enclosed in nested XOR blocks as depicted in the upper part of Fig. 21. When mapping a process schema with loops to a CSTN, we take advantage of this property. Particularly, we first transform each loop into a corresponding XOR structure consisting of two iterations of the respective loop as illustrated

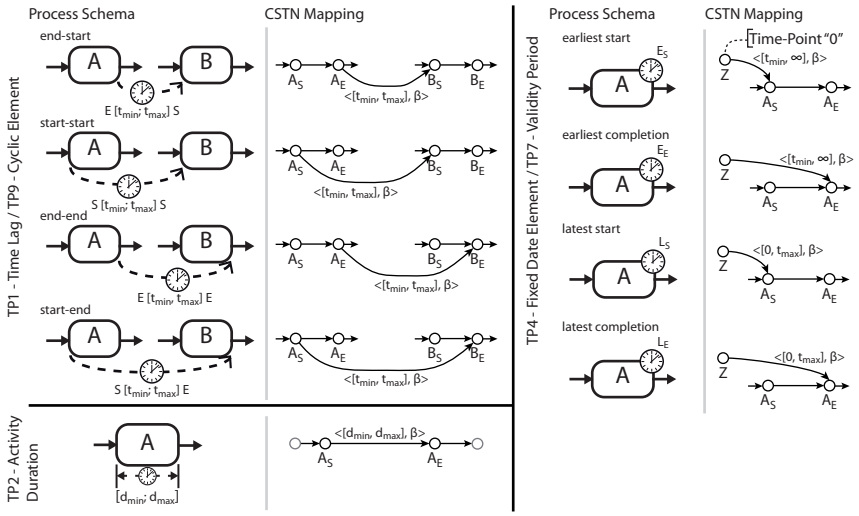

Figure 22: Time Patterns and their mapping to CSTN

in Fig. 21. The second iteration is then followed by a virtual node with a possibly infinite maximum duration representing the case that three or more iterations of the loop are performed. The mapping of this structure is then added to the CSTN (cf. Fig. 21). After completing the second iteration at run-time, if necessary, the virtual node is replaced by another clone of the loop body followed by a new virtual node representing the next iteration. This is repeated for each iteration.

Temporal constraints (i.e., occurrences of the time patterns) are mapped to the CSTN as depicted in Fig. 22. Regarding an activity *Duration* (TP2, *Design Choice C[a]*), if the parameter value of the pattern occurrence is known at build-time (*Design Choice A[a]*), the constraint $\langle [0, \infty]_{A_{iS} A_{iE}}, \beta \rangle$ between the start time-point A_{iS} and the end time-point A_{iE} of the activity, which is created when transforming the process schema, is replaced by a CSTN constraint representing the actual restriction on the activity duration (i.e., the pattern occurrence). For a minimum duration (*Design Choice D[a]*), the minimum value of the CSTN constraint (i.e., 0) is replaced by the actual minimum duration. Likewise, for a maximum duration (*Design Choice D[b]*), the maximum value of the CSTN constraint is replaced by the actual one. For an interval value (*Design Choice D[c]*), both values are replaced accordingly. Finally, if the parameter values of the activity duration are set at process creation-time or during run-time (*Design Choice A[b, c]*), i.e., not all parameter values are known at build-time, the CSTN constraint resulting from the mapping of the activity is not modified, but will be updated as soon as the respective parameter value becomes available. In turn, as soon as all parameter values are known (i.e., either at process creation-time or at some point during run-time), the CSTN constraint is updated according to the actual parameter value. To verify that this mapping indeed reflects the semantics of TP2 (cf. Pattern Semantics 4) reconsider function compareR(φ^t, ψ^t, d) (cf. Def. 8), which constitutes the fundamental part of Pattern Semantics 4. For example, consider *Design Choice D[c]*: According to Pattern Semantics 4 and Def. 8, for event occurrences φ and ψ of the start and end event of the activity, it must hold: $\varphi^t + \min(d) \leq \psi^t \leq \varphi^t + \max(d)$, i.e., $\min(d) \leq \psi^t - \varphi^t \leq \max(d)$. In turn, according to Def. 13, this is exactly the restriction expressed by the corresponding constraint between the two time-points in the CSTN (cf. Fig. 22 and Def. 13). Moreover, as can be easily verified, any restriction regarding valid pairs of events—made by Pattern Semantics 4—is already covered by the applied loop transformation. When mapping a process *Duration* (TP2, *Design Choice C[c]*) to the CSTN, a new constraint reflecting the respective pattern occurrence needs to be added between the time-point representing the start event N_{0S} of the first and the one representing the end event N_{kE} of the last node of the process schema. Besides this, the same considerations as for an activity duration apply. Again, it can be easily verified that this mapping corresponds to Pattern Semantics 4.

A *Time Lag between two Activities* (TP1; Pattern Semantics 1) can be mapped to a CSTN constraint

between the time-points corresponding to the start/end events of the two activities. In particular, depending on the kind of time lag (i.e., start-start, start-end, end-start, end-end; *Design Choice D*), a CSTN constraint is added between the start/end time-point of the source and the start/end time-point of the target activity (cf. Fig. 22). The value of the constraint is then set according to the actual value of the pattern occurrence. If the time lag is a minimum one, only the minimum value of the constraint is set, i.e., the maximum value of the CSTN constraint is set to ∞, indicating that no particular restriction is given. In turn, for a maximum time lag, the minimum value of the respective constraint is set to $-\infty$. When mapping a time lag to the CSTN, particular attention has to be paid to the behaviour of time lags in connection with loops (cf. Pattern Semantics 1); e.g., for a time lag entering a loop the respective CSTN constraint must be added only for the first clone of the respective target activity. Again, Pattern Semantics 1 is based on Def. 8 and hence this mapping reflects the semantics of TP1. Any restrictions regarding valid pairs of events—made by Pattern Semantics 1—is covered by the proper mapping of the time lags entering or exiting a loop. Note that *Time Lags between Arbitrary Events* (TP3; Pattern Semantics 2) could be implemented similarly to TP1. However, AristaFlow (and hence ATAPIS) currently does not provide support for arbitrary events.

For implementing time patterns referring to an absolute point in time (i.e., *Fixed Date Element* and *Validity Period*) one can observe that an absolute point in time may be also represented as the relative time distance between the start of the respective calendar (i.e., time point zero) and the particular point in time. Thus, we add a special time-point Z to the CSTN representing time point zero. A *Validity Period* (TP7) can then be mapped to a CSTN constraint between Z and the respective time-point of the activity as illustrated in Fig. 22. By comparing Defs. 9 and 13, it is easy to verify that this mapping provides the same semantics as TP7.

By definition, the parameter value of a *Fixed Date Element* (TP4) is specific to a process instance and hence is not known before creating this instance. To indicate the presence of a *Fixed Date Element*, at build-time, an unrestricted CSTN constraint with value $\langle[0, \infty], \beta\rangle$ is added between time-point Z and the respective time-point of the activity (process) (cf. Fig. 22). At process creation-time or during run-time, this constraint is then updated according to the actual parameter values of the *Fixed Date Element*. Note that this mapping is similar to the one of a *Validity Period*. Considering the similarity of Pattern Semantics 6, thus, it is easy to verify that this mapping provides the same semantics as Pattern Semantics 5.

A *Schedule Restricted Element* (TP5) cannot be verified at build-time with reasonable effort. To prepare the CSTN for execution, however, at build-time the *Schedule Restricted Element* is mapped to an unrestricted CSTN constraint (i.e., $\langle[0, \infty], \beta\rangle$) between time-point Z and the respective time-point of the activity (or process; *Design Choice D*). Finally, a *Time-Dependent Variability* referring to the execution time of an activity (TP8; *Design Choice J[a]*) is equivalent to an XOR-split whose decision rule is based on the execution time of the activity. Accordingly, the mapping corresponds to the one of an XOR-split.

Given the described mapping of a process schema with temporal constraints, a scenario of a CSTN is similar to an execution trace of the process schema (cf. Def. 2). Further, each solution of a CSTN corresponds to a temporally compliant trace of the process schema (cf. Def. 7). Based on the described mapping, therefore, the temporal consistency of a process schema at build-time can be defined as follows.

Definition 16 (Build-Time Temporal Consistency). *A process schema with temporal constraints is denoted as* build-time temporally consistent *iff the corresponding CSTN is weakly consistent.*

Checking temporal consistency of a process schema can be based on existing CSTN algorithms [55], which are known to be sound and complete. Note that these algorithms can derive all interdependencies between different time-points of the CSTN; i.e., they are able to derive all interdependencies between nodes in the process schema. The result is called a *minimal network*. We take advantage of this minimal network when executing the process schema. Fig. 23 depicts the ATAPIS process editor displaying the CSTN for the process schema from Fig. 18 (left) and the corresponding minimal network (right).

5.3. Checking Temporal Consistency at Run-Time

Temporal consistency of a process schema must be checked during run-time as well. Moreover, the CSTN of the process schema needs to be updated according to the actual execution times of already completed

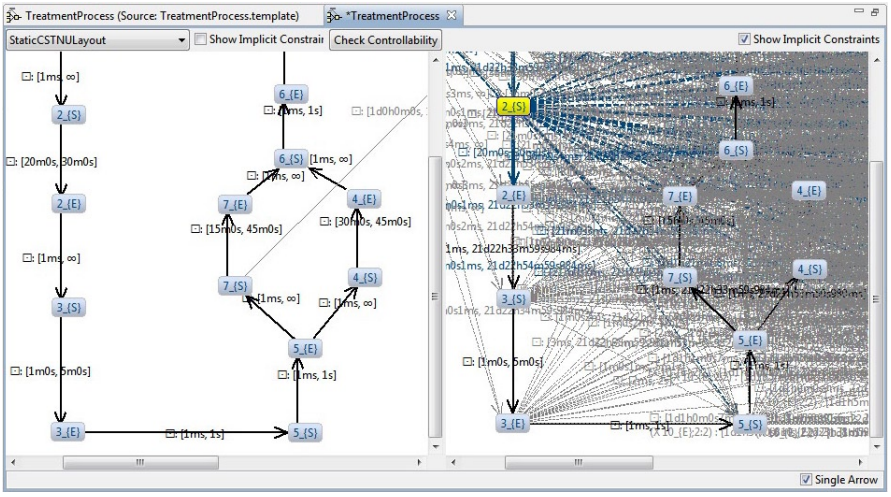

Figure 23: The ATAPIS Process Editor – CSTN and minimal network

activities, to the current execution path, and to the actual parameter values of the temporal constraints [52]. Finally, time patterns that cannot be checked at build-time (e.g., *Schedule Restricted Elements*) can now be verified based on the actual execution time frame of the process instance.

When creating a process instance, the minimal network derived at build-time is cloned. The network is then updated according to the starting time of the process instance. Subsequently, all temporal constraints whose parameter values become known at process creation-time (*Design Choice A[b]*) are considered. Regarding a *Fixed Date Element*, for example, the CSTN constraint created at build-time is updated to the actual parameter values of the pattern occurrence. Moreover, *Schedule Restricted Elements* are considered by restricting the corresponding CSTN constraint such that its bounds are within the first and last possible time slot of the respective schedule. During execution of the process schema, this constraint is updated each time the execution time frame of the respective activity changes.

After updating the minimal network, a consistency check is performed to update the interdependencies between different time-points according to the updated constraints. This ensures that the process schema is still temporally consistent and determines the time frame for starting the first activity. In turn, this start time frame is used by the execution engine to schedule and monitor the start of this activity.

Note that ATAPIS 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 interdependencies between the different nodes, as derived by the CSTN 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 starting the execution of an activity, all temporal constraints related to its start are re-checked to detect possible violations. Subsequently, the minimal network is updated to reflect the actual starting time of that activity. In turn, after completing an activity, its real duration becomes known. Moreover, the parameter values of certain constraints (e.g., *Fixed Date Elements*) might have been set by the just completed activity. In this case, the minimal network is updated accordingly. Then, the consistency checking algorithm is executed to update the interdependencies between nodes as well as the start time frames of the remaining activities if required. Note that this approach ensures that a process instance can only violate one of its temporal constraints if (a) an activity is not started within its time frame, (b) an activity takes longer than permitted by the minimal network, or (c) one of the parameter values set during run-time is not consistent with the process instance. Note that none of these reasons can be controlled by the PAIS. Therefore, as soon as a violation is foreseeable, a time-specific exception handling (i.e., escalation) is triggered.

When executing an XOR-split (i.e., normal XOR-split or *Time-dependent Variability*), a decision regarding the possible execution paths is made, i.e., the truth-value of the literal associated with the corresponding

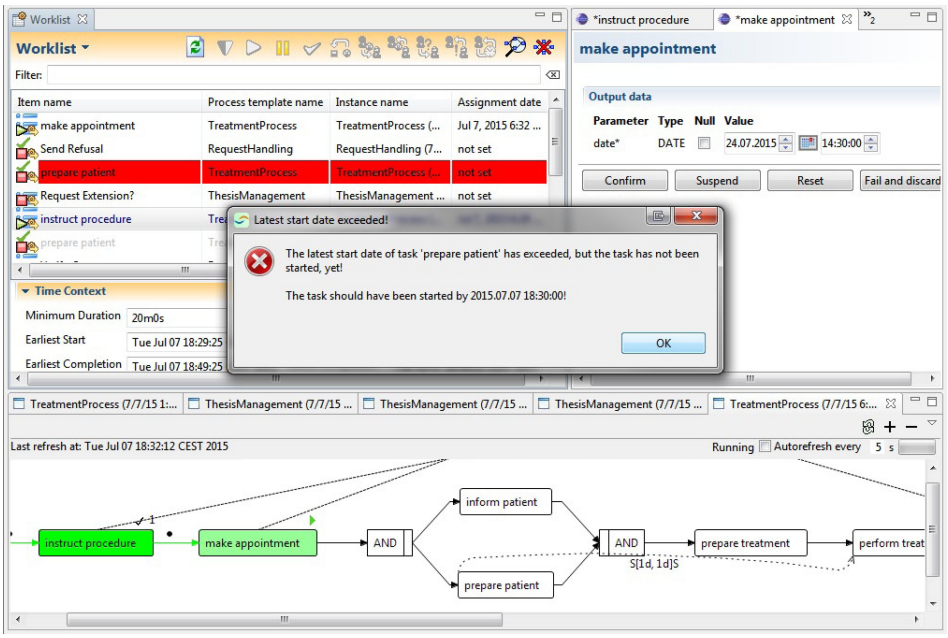

Figure 24: ATAPIS Client

observation nodes becomes known. Therefore, the execution *scenario* is updated and all nodes/edges whose label can no longer be satisfied (i.e., it evaluates to `false`) are removed from the minimal network of the process instance, i.e., skipped XOR branches are removed from the minimal network.

Fig. 24 depicts the ATAPIS client. In its upper left part, the worklist of the currently logged-on user is shown. Below it, the start and completion time frame of activity `instruct procedure`, which is currently selected in the worklist, can be found. The left part shows activity `make appointment`, which is currently being executed by the user. The process instance this activity belongs to is shown in the lower part of Fig. 24. Finally, the system detected that activity `prepare patient` exceeded its latest start date as indicated by the error message. In particular, it was detected that the *Time Lag* between activities `prepare patient` and `perform treatment` and the *Fixed Date* on activity `perform treatment` can no longer be both satisfied.

5.4. Evaluation

To test the implementation and to evaluate the formal semantics of the time patterns, we used ATAPIS to model and simulate selected scenarios originating from our data sources. We selected processes that comprise multiple temporal constraints, that might interact with each other. Moreover, we paid attention that all implemented pattern variants are covered by the selected processes. We then used the resulting process schemas to manually simulate the execution of multiple process instances using different time settings. We simulated both temporally compliant and temporally non-compliant process instances. Then, we used the resulting execution log of each process instance in connection with its process schema and temporal constraints, to test the implementation as well as to evaluate the formal semantics of the time patterns.

First, we checked each process instance (i.e., its execution log) against the process description from the data source. Without knowing the outcome of the implementation, we used the informal description of the process to decide whether or not a particular process instance is temporally compliant. When comparing our assessment with the outcome of the implementation, we were able to confirm that the implementation provides the same classification we obtained. Hence, it matches the semantics of the time patterns. Moreover, it became apparent that in most cases the implementation not only allows detecting violations, but is also able to predict future violations of temporal constraints based on the formal pattern semantics; i.e., the implementation is able to detect whether a process instance can no longer be temporally consistent, even before any temporal constraint is actually violated.

Second, we checked the execution log of each process instance against the formal semantics of the involved time patterns. Again, we could observe the same constraint violations in the execution log, the implementation had already detected.

Altogether, the implementation correctly reflects the formal semantics of the time patterns. Moreover, it indicates that the formal semantics correctly captures the actual semantics of the respective time patterns.

6. Discussion

This section examines features and discusses benefits of defining a formal semantics in general and the presented pattern semantics in particular. In addition, limitations of the latter are discussed and future research directions are presented.

Generally, defining a formal semantics of time patterns provides significant advantages. First, it gives the opportunity to uncover aspects that might have been neglected otherwise as outlined in Examples 14 and 15.

Example 14 (Loops and Time Lags between Activities). *The semantics of the time patterns need to be valid in connection with loops as well. Especially, this applies to TP1 (Time Lags between Activities), i.e., the semantics of a time lag between two activities of which one resides inside a loop and the other one outside that loop is—at first glance—unclear. Actually, four different semantics are conceivable (cf. Sect. 4.2). When defining the semantics for pattern TP1 our aim was to not introduce any ambiguity. Therefore, we chose that semantics, which—in all possible cases—results in the same interpretation for every option of Design Choice E (i.e., minimum, maximum and interval time lag). This semantics covers all cases from our data sources. Moreover, according to our experience, it is the one most obvious to process experts (cf. Sect. 4.2).*

Example 15 (Time-based Restrictions and Comparison of Intervals). *When specifying the formal semantics of the Time-based Restrictions pattern (TP6), we had to define how many instances of an activity exist within a given time frame. In this context, it was required to compare two different time intervals. However, there exist different ways to accomplish this task (e.g., subset, intersection), which have not been considered by the original design choices of this time pattern. To correctly define the semantics of TP6 we added Design Choice U (cf. Fig. 14).*

If such issues are not made explicit and resolved through the definition of a precise, formal semantics, interpretations by PAIS engineers or process designers might be different leading to misunderstandings, ambiguities, or erroneous process implementations (Requirement R1). However, we do not claim to provide the proper semantics for all cases. For example, certain time patterns might have to be interpreted stricter in specific scenarios, whereas they can be considered less restrictive in others. Therefore, decisions had to be made regarding the definition of the formal semantics, which might not suit all possible scenarios (cf. Example 14). On one hand, this might be considered as a limitation of the current pattern semantics definitions. On the other, it is neither possible to completely avoid such decisions nor is it always desirable. In any case, we have exposed the decisions made and hence have made them traceable. Moreover, the presented pattern semantics covers all pattern occurrences we discovered in the extensive data sources. Hence, we may conclude that the decisions made are reasonable.

The defined formal semantics enables us to precisely answer open questions related to the interactions between the time and control flow perspectives as well as the interpretation of certain combinations of process elements (cf. Requirements R2 and R3). For example, if the parameter value of a time pattern may be modified during run-time, it may be unclear which of these values shall be considered for a particular pattern instance (especially in connection with concurrency and loops). By utilizing the defined pattern semantics, respective issues can be precisely addressed.

Based on the defined pattern semantics it becomes possible to compare and evaluate process modeling languages and PAISs according to formally defined criteria. This is further supported by the fact that the provided formal semantics is independent of any process modeling language or paradigm (Requirement R5). Hence, it becomes possible to compare different PAIS and languages regarding their support of the

temporal perspective (e.g., to identify the most suitable one in a given application context). Additionally, the formal semantics provide a foundation for reasoning about modeling language specifications and deriving consequences from them. In turn, the latter is an important aid in validating and evaluating proposed implementations. A formal semantics further provides a view of modeling languages that abstracts from unimportant details of syntax and presentation. In particular, this makes it possible to compare languages as well as to identify language elements additionally required.

The provision of a precise and formal semantics further allows implementing techniques for checking conformance of process instances in respect to a process schema and its temporal constraints. Based on this, one can decide whether the log of a process instance conforms with a given process schema [22]. This is fostered by the use of *temporal execution traces* as basis for formalizing pattern semantics. Hence, it becomes possible to implement conformance checking algorithms based on the defined formal semantics of the time patterns.

Similarly, precise, formal semantics of the time patterns are a key requirement for formally verifying of the temporal perspective of processes at build-time as well as for verifying and monitoring it during run-time. In particular, the formal semantics allow solving open issues when developing with verification procedures for processes comprising temporal constraints (cf. Example 14 and Sect. 5). Especially, this applies to more complex time patterns as well as to the link between time patterns and loops. So far, respective issues have been "avoided" by excluding both complex time patterns and loops from respective considerations. However, without considering these constructs no complete specification of the temporal perspective would be possible (Requirement R2). In turn, without a formal basis no universally valid verification of the temporal perspective can be accomplished due to ambiguities and unclear semantics.

A limitation of the presented formal semantics and an avenue for future work is the fact that the formal semantics neither consider build- nor run-time support of the temporal perspective. At build-time the consistency of a process schema must be verified to ensure that corresponding process instances can be executed without violating any temporal constraint. In turn, when executing process instances with temporal constraints, challenging run-time issues emerge like "How can temporal constraints be enforced?" or "What happens if a temporal constraint has been violated or is in danger of being violated?". Moreover, one must differentiate between temporal constraints to be met in any case (e.g., delivery deadlines) and constraints representing a recommendation that may be ignored (e.g., duration constraints). Furthermore, in certain cases, activity *Durations* may need to be restricted to ensure that a process instance can be completed without violating a constraint, whereas in other cases it must be ensured that the entire range of the activity *Duration* is available for executing the activity [52]. Although such issues are out of the scope of this paper, the presented formal pattern semantics serve as a good basis for exploring them. To the best of our knowledge, this paper provides the first attempt to precisely and formally define the semantics of all the time patterns. Finally, we provide an implementation of selected time patterns tackling some of the open questions.

Regarding the run-time support of the temporal perspective, as a next step a comprehensive *execution semantics* for process instances with temporal constraints needs to be defined. Such an execution semantics not only needs to consider the different styles of temporal constraints (i.e., optional vs. mandatory), but also define at which points during run-time a particular constraint should be (re-)evaluated and the impact this re-evaluation might have on process execution. Further, we need to specify what happens if a particular constraint is violated (e.g., escalation, exception handling). These issues are still part of ongoing research and are thus out of scope of this paper. Note that the provided implementation constitutes a first step towards such a comprehensive execution semantics.

7. Threats to Validity

This section discusses threats regarding the validity of our work and how we dealt with them. The main threats include *inaccuracy in specifying pattern semantics*, the *identification procedure* of the pattern semantics, their *incomplete implementation* and *completeness* of the patterns and their semantics.

First, formal semantics always carry the risk of being inaccurate. This threat is twofold: On one hand, there exists the risk that the formal semantics exclude valid cases. In turn, this bears the risk of invalidating

the patterns themselves due to a too restrictive semantics. On the other hand, the formal semantics may allow for invalid cases, rendering the formal semantics useless. To alleviate this threat, we analyzed a large set of data sources with more than 400 processes. When identifying and formalizing the pattern semantics, we followed a thorough design science approach [23] to ensure validity and reliability of the results. In particular, after formalizing the pattern semantics, we re-evaluated each of the initially identified pattern occurrences to ensure its coverage by the formal pattern semantics. Likewise we have ensured that the semantics indeed disallow for all invalid cases of the respective pattern occurrence. Hence, we are confident that the formal pattern semantics are accurate. Nevertheless, it constitutes just one possible way of describing the semantics of the time patterns, i.e., there may be others being correct as well. Considering our experience in the field of PAISs and temporal constraint handling, however, we strongly believe that the described semantics not only make sense but are correct as well. This has been underpinned by the presented implementation of the time patterns based on their formal semantics.

Second, both the identification and formalization of the pattern semantics were carried out by the main author. This might comprise subjective decisions since several of the evaluated pattern occurrences did not provide a clear description of the respective temporal constraint. Moreover, the data sources might already contain subjective decisions made by their creators. To mitigate this risk, we applied a rigorous extraction and re-evaluation procedure. In addition, the co-authors continuously cross-checked the results of the first author. Any disagreements were resolved by a consensus-building process [58] including discussions and in-depth analyses of the respective pattern occurrences. To alleviate the risk of working with biased data sources, the latter were obtained from different sources.

Third, not all variants of the time patterns were implemented. This threatens validity in so far as the semantics of the time pattern variants have not been completely verified through an implementation. Moreover, it raises the question whether the non-implemented pattern variants and their semantics can indeed be implemented. The answer to this question is twofold. First, we are aware that it might not be possible to implement all variants of the time patterns in a general-purpose PAIS. For example, time patterns referring to a set of process instances, such as *Durations* of a set of process instances (TP2 *Design Choice C[d]*) or *Time-based Restrictions* for a set of activities from different process instances (TP6 *Design Choice G[b, c]*), are hard to implement in a general way. Particularly, it is not clear how to identify and relate respective process instances. Nevertheless, the data sources have confirmed that respective pattern variants as well as their formal semantics are both useful and required in practice. In particular, these time patterns are still valuable when using patterns for requirements elicitation or implementing respective processes in terms of a traditional software system. Note that for a specific application scenario it will still be possible to manually implement relevant pattern variants based on the proposed pattern semantics. Furthermore, the knowledge about the patterns and their semantics helps decision makers in deciding whether to use a general-purpose PAIS or to develop a specifically tailored software system. Finally, the threat of not having tested all pattern variants through an implementation is mitigated by the rigorous re-evaluation process performed after pattern identification and formalization. Thus, even without an implementation of all pattern variants, we are confident that the pattern semantics are valid.

Fourth, the completeness of the time patterns constitutes another issue. The latter is out of the scope of this paper as we do not aim to identify further time patterns. Apart from this, completeness constitutes an inherent problem of any work that aims to identify evidenced patterns from real world cases. Nevertheless, this might threaten the validity of our work. Due to the large set of data sources we analyzed as well as the thorough systematic literature review performed when eliciting the time patterns [18], we are confident that the identified time pattern set is representative. Moreover, this threat is further mitigated by the fact that the semantics of the various time patterns are defined independently of each other. Finally, as time pattern semantics are based on a generic formalism, it will pose no problem to add the semantics of additional time pattern; i.e., extensibility of the pattern set and pattern semantics, respectively, is ensured.

8. Related Work

In [17], we informally introduced 10 time patterns. In turn, in [18] we provided more detailed descriptions and validated the time patterns through a systematic literature review and an evaluation of selected approaches

from academia and industry regarding their support of the time patterns. Since their introduction, the time patterns were picked up by other groups in different context. For example, Barba et al. [59] use them in the context of providing scheduling support for declarative workflows. In turn, Döhring and Zimmermann [60, 61] apply the time patterns in combination with workflow adaptation patterns. In [62], time stream Petri net (TSPN) representations for selected time patterns are proposed in order to demonstrate that TSPN is a suitable formalism for supporting the life cycle of processes with temporal constraints. Lenhard et al. [63, 64] develop a framework for evaluating the degree of support a (process modeling) language provides for different kinds of patterns (including the time patterns). The authors apply this framework to assess service orchestration languages (e.g., BPEL, SunBPEL, and WF4). They conclude that "a better formalization of several pattern catalogs in terms of execution traces would be beneficial" [64]. In the past, no formal semantics for the time patterns were provided, even though such a rigorous basis is crucial when implementing and comparing time-aware PAISs. This paper closes this gap.

The general idea of using patterns to compare PAISs has been proposed by the workflow patterns project [9]. Based on patterns covering different perspectives, the expressiveness of process meta models and process modeling tools can be assessed. Additional patterns were introduced, including data flow patterns [10], resource patterns [11], exception handling patterns [14], activity patterns [13], user interface transformation patterns [65], and process change patterns [12, 16]. Several of them come with formal semantics, e.g., based on languages such as pi-calculus [39] or execution traces [16].

Considerable work on temporal process constraints [19, 5, 20, 7, 66] exists. Most approaches focus on modeling and verification issues. Bettini et al. [19] use Simple Temporal Networks (STN), a special variant of temporal constraint networks [57], as the basic formalism for representing and reasoning about temporal process constraints. According to the approach suggested by [19], each activity is represented by two nodes in an STN, representing the starting and ending time point of the activity. In turn, STN edges represent temporal constraints and precedence relations between the corresponding nodes. Combi et al. [5, 20] use STN as formalism for verifying the consistency of the temporal perspective of a process schema. The transformation process is similar to the one of Bettini et al. In addition, well-nested loops are considered. Thereby, each loop is associated either with a minimum and maximum number of iterations or a maximum duration for completing all loop iterations. A loop can then be transformed into a consecutive set of XOR-splits for deciding whether the next iteration shall be executed or remaining iterations be skipped (cf. Sect. 5). Eder et al. [7] use Timed Workflow Graphs, which constitute an extension of the Critical Path Method (CPM) [67]—a well known project planning method—to represent temporal properties of activities and their control flow relations. Essentially, a timed workflow graph is similar to a process schema. Each activity in the process has a fixed duration and is augmented by two values representing its earliest and latest completion time. Zhuge et al. [66] define temporal constraints in terms of restrictions of the start and completion times of activities. Note that this is similar to the way the semantics presented in this paper are defined. Moreover, different time zones are considered when defining temporal constraints, although this is not really required. On one hand, temporal constraints are usually specified with respect to a reference time zone; on the other, temporal constraints can be converted from one time zone to another [66].

To evaluate how well the presented formal pattern semantics match the ones inherent to these approaches, we informally compared them. The approaches were selected since they provide the broadest support of the time patterns and provide at least some kind of semi-formal description of the temporal constraints supported. Note that only a semi-formal comparison of the formal pattern semantics with these approaches is possible as for none of the latter either a rigorous formal description or a reference implementation is available. For the sake of brevity, we do not discuss our findings in detail in this paper. Interested readers are referred to a technical report discussing this comparison [68]. Main results are summarized in Table 4.

Other approaches dealing with the temporal perspective of PAISs focus on specific time support features like escalation management [69] and scheduling support [59, 70]. In turn, [71] investigates the mutual dependencies among different temporal constraints and derives inference rules for their verification and monitoring. Additionally, time support features like process monitoring [72], process mining [73], and the effects of process changes on temporal constraints [74] have been studied. A systematic elaboration of the requirements for time support as well as a respective formal semantics has been missing so far.

Our implementation uses CSTN [55]–a special kind of *Temporal Constraint Satisfaction Problem* [57]–to

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		∘	?	∘	∘

[a] Does not consider loops.

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

(+)	Equivalent to a restricted variant of the pattern semantics.	?	Discussed but no implementation is provided.
(∗)	Not discussed/No implementation provided, but may be implemented with a semantics equivalent to a restricted variant of the presented pattern semantics.	–	Different semantics (cf. [68] for details).
		∘	Not considered.

Table 4: Assessment of Different Approaches

verify the satisfiability of the temporal constraints of a process schema (i.e., its temporal consistency) according to the time patterns semantics. However, other techniques for verifying real-time systems could be used as well. In particular, there exists a variety of temporal constraint satisfaction problems like CSTNU [56] and CTPP [75]. In turn, [76, 77] use timed automata [42] in combination with model checking techniques like Metric Temporal Logic (MTL) [78] to verify the satisfiability of a process schema with temporal constraints. In this context, [79] shows how CSTNs can be translated into *Timed Game Automata* and then verified using model checking tools like UPPAAL-Tiga [80]. In [81] a framework for the formal verification of real-time UML statecharts based on hierarchical timed automata and model checking is presented. Finally, [62] presents time stream Petri net (TSPN) representations for selected time pattern variants. In theory, all these techniques may be used for verifying the satisfiability of a process schema with temporal constraints. However, existing approaches only partially cover the time patterns and their variants. For implementing the time patterns, we have chosen CSTNs as they are similar to the temporal execution traces used for defining the formal pattern semantics, and at the same time have a lower computational complexity compared to other techniques.

9. Summary and Outlook

We have specified the formal semantics of the time patterns we originally introduced in [17, 18]. Their initial introduction complemented existing workflow patterns. In particular, time patterns allow for a more meaningful evaluation of PAISs if the handling of temporal constraints becomes crucial. In combination with workflow patterns, time patterns enable PAIS engineers to choose the PAIS-enabling technology meeting their requirements best. The formal semantics presented in this paper provide the basis for implementing the patterns in PAISs as well as for comparing PAISs with respect to their support of the temporal perspective. For each pattern, its formal semantics was specified in a language-independent manner based on traces.

Our future work will consider time patterns for aspects other than control flow as well. Further, we will conduct a comprehensive study on time support features like the verification of temporal constraints, escalation management, and scheduling support. In this context, we are currently developing a comprehensive execution semantics for process instances with temporal constraints.

References

[1] M. Reichert, B. Weber, Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies, Springer, 2012.

[2] B. Weber, M. Reichert, W. Wild, S. Rinderle-Ma, Providing integrated life cycle support in process-aware information systems, Int'l J Cooperative Information Systems 18 (1) (2009) 115–165.

[3] R. Lenz, M. Reichert, IT support for healthcare processes - premises, challenges, perspectives, Data & Knowledge Engineering 61 (1) (2007) 39–58.

[4] D. Müller, J. Herbst, M. Hammori, M. Reichert, IT support for release management processes in the automotive industry, in: Proc 4th Int'l Conf Business Process Management (BPM'06), Vol. 4102 of LNCS, 2006, pp. 368–377.

[5] C. Combi, M. Gozzi, J. M. Juarez, B. Oliboni, G. Pozzi, Conceptual modeling of temporal clinical workflows, in: Proc 14th Int'l Symp on Temporal Representation and Reasoning (TIME'07), 2007, pp. 70–81.

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

[7] J. Eder, E. Panagos, M. Rabinovich, Time constraints in workflow systems, in: Proc 11th Int'l Conf Advanced Information Systems Engineering (CAiSE'99), Vol. 1626 of LNCS, 1999, pp. 286–300.

[8] P. Dadam, M. Reichert, K. Kuhn, Clinical workflows - the killer application for process-oriented information systems., in: Proc 4th Int'l Conf Business Information Systems (BIS'00), 2000, pp. 36–59.

[9] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, A. P. Barros, Workflow patterns, Distributed and Parallel Databases 14 (1) (2003) 5–51.

[10] N. C. Russell, A. H. M. ter Hofstede, D. Edmond, W. M. P. van der Aalst, Workflow data patterns: Identification, representation and tool support, in: Proc 24th Int'l Conf Conceptual Modeling, Vol. 3716 of LNCS, 2005, pp. 353–368.

[11] N. C. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, D. Edmond, Workflow resource patterns: Identification, representation and tool support, in: Proc 17th Int'l Conf Advanced Information Systems Engineering (CAiSE'05), Vol. 3520 of LNCS, 2005, pp. 216–232.

[12] B. Weber, M. Reichert, S. Rinderle-Ma, Change patterns and change support features - enhancing flexibility in process-aware information systems, Data & Knowledge Engineering 66 (3) (2008) 438–466.

[13] L. Thom, M. Reichert, C. Iochpe, Activity patterns in process-aware information systems: Basic concepts and empirical evidence, Int'l J Business Process Integration & Management 4 (2) (2009) 93–110.

[14] N. C. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, Exception handling patterns in process-aware information systems, Tech. rep., BPMCenter.org (2006).

[15] B. Weber, S. Rinderle, M. Reichert, Change patterns and change support features in process-aware information systems, in: Proc 19th Int'l Conf Advanced Information Systems Engineering (CAiSE'07), Vol. 4495 of LNCS, 2007, pp. 574–588.

[16] S. Rinderle-Ma, M. Reichert, B. Weber, On the formal semantics of change patterns in process-aware information systems, in: Proc 27th Int'l Conf Conceptual Modeling (ER'08), Vol. 5231 of LNCS, 2008, pp. 279–293.

[17] A. Lanz, B. Weber, M. Reichert, Workflow time patterns for process-aware information systems, in: Proc 11th Int'l Workshop BPMDS 2010, Vol. 50 of LNBIP, 2010, pp. 94–107.

[18] A. Lanz, B. Weber, M. Reichert, Time patterns for process-aware information systems, Requirements Engineering 19 (2) (2014) 113–141.

[19] C. Bettini, X. S. Wang, S. Jajodia, Temporal reasoning in workflow systems, Distributed and Parallel Databases 11 (3) (2002) 269–306.

[20] C. Combi, M. Gozzi, R. Posenato, G. Pozzi, Conceptual modeling of flexible temporal workflows, ACM Transactions on Autonomous and Adaptive Systems 7 (2) (2012) 19:1–19:29.

[21] R. van Glabbeek, U. Goltz, Refinement of actions and equivalence notions for concurrent systems, Acta Informatica 37 (2001) 229–327.

[22] A. Rozinat, W. M. P. van der Aalst, Conformance checking of processes based on monitoring real behavior, Information Systems 33 (1) (2008) 64–95.

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

[24] A. R. Hevner, S. Chatterjee, Design Research in Information Systems, Vol. 22 of Integrated Series in Information Systems, 2010.

[25] B. Schultheiß, J. Meyer, R. Mangold, T. Zemmler, M. Reichert, Prozessentwurf für den Ablauf einer stationären Chemotherapie, Tech. Rep. DBIS-5, Universität Ulm (1996).

[26] B. Schultheiß, J. Meyer, R. Mangold, T. Zemmler, M. Reichert, Prozessentwurf am Beispiel eines Ablaufs einer OP, Tech. Rep. DBIS-6, Universität Ulm (1996).

[27] C. Li, Mining process model variants: Challenges, techniques, examples, Phd thesis, University of Twente (2010).

[28] C. Li, M. Reichert, A. Wombacher, The MinAdept clustering approach for discovering reference process models out of process variants, Int'l J Cooperative Information Systems 19 (3 & 4) (2010) 159–203.

[29] C. Li, M. Reichert, A. Wombacher, Mining business process variants: Challenges, scenarios, algorithms, Data & Knowledge Engineering 70 (5) (2011) 409–434.

[30] R. Bobrik, Konfigurierbare Visualisierung komplexer Prozessmodelle, Ph.D. thesis, University of Ulm (2008).

[31] German Association of the Automotive Industry (VDA), Engineering change management. Part 1: Engineering change request (ECR) (2005).

[32] Federal Aviation Administration, Aviation maintenance technician handbook, chapter 8. inspection fundamental, `http://www.faa.gov/regulations_policies/handbooks_manuals/aircraft/amt_handbook/media/FAA-8083-30_Ch08.pdf` (2008).

[33] IACA (Int'l Air Carrier Association), Subpart Q - flight and duty time limitations and rest requirements, http://www.iaca.be/iaca/library/q15922_3.pdf (2004).
[34] M. Stoicsics, Evaluation des Konzeptes eines Catering-, Steuerungs- und Controllingsystems am Vergleichsbeispiel des Endmontageprozesses der Fertigung von hochisolierenden Lager- und Transportbehältern für das Luftfahrt-Catering, Diploma thesis, University of Ulm (2011).
[35] M. Steinle, Einsatz von Workflow Engines zur effizienten Anpassung standardisierter Branchensoftware an unterschiedliche Kundenanforderungen, Masters thesis, Ulm University (2005).
[36] G. Grambow, R. Oberhauser, M. Reichert, Event-driven exception handling for software engineering processes, in: Proc 5th Int'l Workshop on Event-Driven Business Process Management (edBPM'11), Vol. 99 of LNBIP, 2012, pp. 414–426.
[37] A. Hallerbach, Management von Prozessvarianten, Phd thesis, University of Ulm (2010).
[38] ISO/EIC, ISO19700-1 Software asset management (2006).
[39] F. Puhlmann, M. Weske, Using the pi-calculus for formalizing workflow patterns, in: Proc 3rd Int'l Conf Business Process Management (BPM'05), Vol. 3649 of LNCS, 2005, pp. 153–168.
[40] M. Huth, M. Ryan, Logic in Computer Science - modelling and reasoning about systems, 2004.
[41] A. Cerone, A. Maggiolo-Schettini, Time-based expressivity of time petri nets for system specification, Theoretical Computer Science 216 (1-2) (1999) 1 – 53.
[42] R. Alur, D. Dill, A theory of timed automata, Theoretical Computer Science 126 (2) (1994) 183 – 235.
[43] A. Lanz, M. Reichert, Enabling time-aware process support with the atapis toolset, in: Proc. BPM'14 Demo Track, 2014.
[44] B. Kiepuszewski, A. H. M. ter Hofstede, C. Bussler, On structured workflow modelling, in: Proc 12th Int'l Conf Advanced Information Systems Engineering (CAiSE'00), Vol. 1789 of LNCS, 2000, pp. 431–445.
[45] J. Mendling, H. A. Reijers, W. M. P. van der Aalst, Seven process modeling guidelines (7PMG), Information and Software Technology 52 (2) (2010) 127–136.
[46] J. Vanhatalo, H. Völzer, F. Leymann, Faster and more focused control-flow analysis for business process models through sese decomposition, in: Proc 5th Int'l Conf Service-Oriented Computing (ICSOC'07), Vol. 4749 of LNCS, 2007, pp. 43–55.
[47] S. Rinderle, M. Reichert, P. Dadam, Flexible support of team processes by adaptive workflow systems, Distributed and Parallel Databases 16 (1) (2004) 91–116.
[48] P. Terenziani, Integrating calendar dates and qualitative temporal constraints in the treatment of periodic events, IEEE Transactions on Knowledge and Data Engineering 9 (5) (1997) 763–783.
[49] L. Anselma, Recursive representation of periodicity and temporal reasoning, in: Proc 11th Int'l Symp on Temporal Representation and Reasoning (TIME'04), 2004, pp. 52–59.
[50] P. Dadam, et al., From ADEPT to AristaFlow BPM Suite: A research vision has become reality, in: Proc Business Process Management Workshops, Vol. 43 of LNBIP, 2009, pp. 529–531.
[51] A. Lanz, M. Reichert, Dealing with changes of time-aware processes, in: Proc Business Process Management (BPM'14), Vol. 8659 of LNCS, 2014, pp. 217–233.
[52] A. Lanz, R. Posenato, C. Combi, M. Reichert, Controllability of time-aware processes at run time, in: Proc 21st Int'l Conf Cooperative Information Systems (CoopIS'13), no. 8185 in LNCS, 2013, pp. 39–56.
[53] M. Reichert, P. Dadam, ADEPTflex – supporting dynamic changes of workflows without losing control, J Intelligent Information Systems 10 (2) (1998) 93–129.
[54] B. Leban, D. McDonald, D. Forster, A representation for collections of temporal intervals, in: Proc 5th Int'l Conf Artificial Intelligence (AAAI'86), Vol. 86, 1986, pp. 367–371.
[55] I. Tsamardinos, T. Vidal, M. Pollack, CTP: A new constraint-based formalism for conditional, temporal planning, Constraints 8 (4) (2003) 365–388.
[56] L. Hunsberger, R. Posenato, C. Combi, The dynamic controllability of conditional STNs with uncertainty, in: Proc Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx), 2012.
[57] R. Dechter, I. Meiri, J. Pearl, Temporal constraint networks, Artificial Intelligence 49 (1991) 61–95.
[58] J. Recker, N. Safrudin, M. Rosemann, How novices model business processes, in: R. Hull, J. Mendling, S. Tai (Eds.), Business Process Management, Vol. 6336 of LNCS, 2010, pp. 29–44.
[59] I. Barba, A. Lanz, B. Weber, M. Reichert, C. del Valle, Optimized time management for declarative workflows, in: Proc 13th Int'l Conf BPMDS 2012, Vol. 113 of LNBIP, 2012, pp. 195–210.
[60] M. Döhring, B. Zimmermann, L. Karg, Flexible workflows at design- and runtime using bpmn2 adaptation patterns, in: W. Abramowicz (Ed.), Business Information Systems, Vol. 87 of LNBIP, 2011, pp. 25–36.
[61] B. Zimmermann, M. Döhring, Patterns for flexible bpmn workflows, in: Proc 16th European Conf Pattern Languages of Programs (EuroPLoP'11), ACM Int'l Conf Proceeding Series, 2012, pp. 7:1–7:9.
[62] F. Cicirelli, A. Furfaro, L. Nigro, Using time stream petri nets for workflow modelling analysis and enactment, Simulation 89 (1) (2013) 68–86.
[63] J. Lenhard, A. Schönberger, G. Wirtz, Edit distance-based pattern support assessment of orchestration languages, in: On the Move to Meaningful Internet Systems: OTM 2011, Vol. 7044 of LNCS, 2011, pp. 137–154.
[64] J. Lenhard, A. Schönberger, G. Wirtz, Streamlining pattern support assessment for service composition languages, in: Proc 3rd Workshop on Services and their Composition (ZEUS'11), Vol. 705 of CEUR Workshop Proc, 2011, pp. 112–119.
[65] J. Kolb, P. Hübner, M. Reichert, Automatically generating and updating user interface components in process-aware information systems, in: Proc 20th Int'l Conf Cooperative Information Systems, no. 7565 in LNCS, 2012, pp. 444–454.
[66] H. Zhuge, T.-Y. Cheung, H.-K. Pung, A timed workflow process model, J Systems and Software 55 (3) (2001) 231–243.
[67] S. Philipose, Operations research - a practical approach (1986).
[68] A. Lanz, M. Reichert, B. Weber, A formal semantics of time patterns for process-aware information systems, Tech. Rep. UIB-2013-02, University of Ulm (2013).

[69] W. M. P. van der Aalst, M. Rosemann, M. Dumas, Deadline-based escalation in process-aware information systems, Decision Support Systems 43 (2) (2007) 492–511.
[70] C. Combi, G. Pozzi, Task scheduling for a temporal workflow management system, in: Proc 13th Int'l Symp on Temporal Representation and Reasoning (TIME'06), 2006, pp. 61–68.
[71] J. Chen, Y. Yang, Temporal dependency based checkpoint selection for dynamic verification of temporal constraints in scientific workflow systems, ACM Transactions on Software Engineering and Methodology 20 (3) (2011) 9:1–9:23.
[72] M. Sayal, F. Casati, U. Dayal, M.-C. Shan, Business process cockpit, in: Proc 28th Int'l Conf Very Large Data Bases (VLDB'02), VLDB Endowment, 2002, pp. 880–883.
[73] W. M. P. van der Aalst, M. Pesic, M. Song, Beyond process mining: From the past to present and future, in: Proc Int'l Conf Advanced Information Systems Engineering (CAiSE'10), Vol. 6051 of LNCS, 2010, pp. 38–52.
[74] S. W. Sadiq, O. Marjanovic, M. E. Orlowska, Managing change and time in dynamic workflow processes, Int'l J Cooperative Information Systems 9 (1-2) (2000) 93–116.
[75] M. Falda, F. Rossi, K. Venable, Strong, weak, and dynamic consistency in fuzzy conditional temporal problems, in: Proc of COPLAS 2007, CP'07 workshop on planning and scheduling, 2007.
[76] E. de Maria, A. Montanari, M. Zantoni, An automaton-based approach to the verification of timed workflow schemas, in: Proc 13th Int'l Symp on Temporal Representation and Reasoning (TIME'06), 2006, pp. 87–94.
[77] F. M. Maggi, M. Westergaard, Using timed automata for a priori warnings and planning for timed declarative process models, Int'l J Cooperative Information Systems 23 (01) (2014) 1440003.
[78] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Systems 2 (4) (1990) 255–299.
[79] A. Cimatti, L. Hunsberger, A. Micheli, R. Posenato, M. Roveri, Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation, in: 21st Int'l Symp Temporal Representation and Reasoning (TIME), 2014, pp. 27–36.
[80] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, D. Lime, Uppaal-tiga: Time for playing games!, in: W. Damm, H. Hermanns (Eds.), Computer Aided Verification, Vol. 4590 of LNCS, 2007, pp. 121–125.
[81] A. David, M. Möller, W. Yi, Formal verification of uml statecharts with real-time extensions, in: Fundamental Approaches to Software Engineering, Vol. 2306 of LNCS, 2002, pp. 218–232.