ulm university universität
uulm

A Formal Semantics of Time Patterns for Process-aware Information Systems

Andreas Lanz, Manfred Reichert, Barbara Weber

Ulmer Informatik-Berichte

Nr. 2013-02
Januar 2013

A Formal Semantics of Time Patterns for Process-aware Information Systems

Andreas Lanz[1], Manfred Reichert[1], and Barbara Weber[2]

[1] Institute of Databases and Information Systems, Ulm University, Germany
{Andreas.Lanz,Manfred.Reichert}@uni-ulm.de
[2] Quality Engineering Research Group, University of Innsbruck, Austria
Barbara.Weber@uibk.ac.at

Summary. Companies increasingly adopt process-aware information systems (PAISs) to coordinate, monitor and evolve their business processes. Although the proper handling of temporal constraints (e.g., deadlines and minimum time lags between activities) is crucial in many application domains, existing PAISs vary significantly regarding their support of the temporal perspective of business processes. Both the formal specification and operational support of temporal constraints constitute fundamental challenges in this context. In previous work, we introduced *time patterns* facilitating the comparison of PAISs in respect to their support of the temporal perspective and provided empirical evidence for them. To avoid ambiguities and to ease the use as well as implementation of the time patterns, this paper formally defines their semantics. To enable pattern use in a wide range of process modeling languages and pattern integration with existing PAISs, this semantics is expressed independent of a particular process meta model. Altogether, the presented pattern formalization will foster the integration of the temporal perspective in PAISs.

Key words: Process-aware Information System, Workflow Patterns, Time Patterns, Temporal Perspective, Temporal Constraints

1 Introduction

Companies strive for improved life cycle support of their business processes [1, 2]. In particular, IT support for analyzing, modeling, executing and monitoring these processes results in competitive advantages [3, 4]. In this context, process-aware information systems (PAISs) offer promising perspectives towards process automation by enabling companies to define a business process in terms of an explicit *process schema* and to execute related *process instances* based on this schema in a controlled and efficient manner [1].

Both the formal specification and operational support of *temporal constraints* constitute fundamental challenges of any PAIS [5, 6, 7, 8]. In contemporary PAISs, however, time support is very limited. Furthermore, there exist no criteria for systematically assessing the degree of support of the temporal perspective across different PAISs and PAIS-enabling technologies (e.g., workflow management systems and case handling tools [9]). To make PAISs better comparable and

facilitate the selection of PAIS-enabling technologies in a given application environment, *workflow patterns* have been introduced [10, 11, 12, 13]. These patterns allow analyzing the expressiveness of process modeling languages and tools in respect to different process perspectives, like, for example, control flow [10], data flow [11], resources [12], activities [14], exceptions [15], and process change [13, 16, 17].

1.1 Problem Statement

Recently, we extended the workflow patterns by a set of 10 *time patterns* suitable for evaluating the support of the temporal perspective in a PAIS [18, 19]. Examples of such time patterns include *Time Lags between Activities*, *Durations*, and *Fixed Date Elements*. Empirical evidence we gained in a number of case studies has confirmed that identified time patterns are common in practice and required for properly modeling the temporal perspective of processes in a variety of application domains [19, 20]. Furthermore, we evaluated different approaches and tools in respect to their time pattern support [19].

Our evaluations have emphasized the need for a precise and formal semantics of the identified time patterns. In particular, ambiguities in respect to this semantics would hamper both pattern implementation and comparison of existing PAISs. So far, the time patterns we identified have lacked a formal semantics. Similar to workflow patterns [21], however, the latter is crucial in order to enable a widespread use of the time patterns, i.e., for each time pattern, its effects on process enactment must be precisely defined. This becomes necessary to ensure that different implementations of the respective pattern have the same basic meaning. Note that this is also required to ensure a common understanding of process schemata containing time patterns. Furthermore, it becomes necessary to specify how the time patterns interact with different elements of the control flow perspective (i.e., control flow patterns like loops, XOR-splits, or AND-joins). Finally, for time patterns depending on run-time data (i.e., process instance data), it must be precisely defined which data value to use, otherwise different implementations might result in different behaviour.

1.2 Contribution

This paper complements our previous work on time patterns [19] by additionally formalizing the semantics of these patterns. This will allow users to ground pattern implementation as well as pattern-based analysis of PAISs on a solid and formal basis. Furthermore, it will foster the widespread use of the time patterns by PAIS engineers and researchers.

We provide a *precise, formal semantics* for the time patterns presented in [19]. This semantics is defined independent of a specific process modeling language or paradigm. Furthermore, we illustrate pattern semantics in terms of examples and detailed explanations. Finally, we validate the proposed pattern semantics to ensure that it meets the one expected by domain experts.

When defining the semantics of the time patterns, a particular challenge is to provide a formal pattern description independent of a particular process modeling language. Only then PAIS engineers will be able to integrate the time patterns without need to cope with language-specific issues, which again might result in ambiguities. To define a pattern semantics independent of any process modeling language and closely related to the execution semantics of processes, we use process *execution traces* as basis for our formalization [22]. In particular, an execution trace can be considered as *execution history* of a process instance, i.e., it records what happened during the execution of a particular process instance. We then formally describe the semantics of time patterns by indicating which traces can be produced on a given process schema containing a particular time pattern, i.e., which traces are *compliant* with pattern semantics. Amongst others, this makes it possible to implement techniques for checking the conformance [23] of a process instance in respect to a process schema and its temporal constraints (i.e., occurrences of the time patterns).

In case a pattern instance additionally depends on run-time data of the process instance we precisely define which data value is valid for a specific pattern instance. In particular, if the same *data object* is modified several times during process execution, we precisely define which of these data values shall be considered for a pattern instance. This is especially important in connection with loops or concurrent data access; otherwise ambiguities will hamper pattern implementation.

Based on the defined semantics, pattern implementation is put on a sound basis. Moreover, ambiguities are avoided regarding the implementation of the patterns in a PAIS as well as their use for assessing existing PAISs. Finally, a precise formal semantics is a prerequisite for verifying the temporal perspective of business processes at both build- and run-time [5, 6, 7, 24, 25]. Overall, the defined semantics will foster the integration of the temporal perspective into existing PAISs and hence broaden their scope significantly.

The remainder of this paper is organized as follows: Section 2 provides background information and recalls the time patterns we informally presented in [19]. Section 3 discusses the research method applied for defining and validating the proposed formal pattern semantics. Section 4 provides the formal semantics of the time patterns. In Section 5, we provide a validation of the formal semantics by evaluating how well it matches the semantics inherent to other academic approaches. In Section 6, we discuss the expected impact of the proposed formal pattern semantics as well as its usefulness for implementing time support in PAISs. Section 7 discusses related work and Section 8 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 [18, 19].

2.1 Basic Notions

A *process-aware information system* (PAIS) is a specific type of information system providing process support functions. It is characterized by the separation of process logic from application code. At build-time, process logic is explicitly defined based on the constructs provided by a *process meta model*. For each business process to be supported, 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 *control connectors* (e.g., Start-/End-Nodes, XOR-splits, and AND-joins) – and a set of *control edges* linking these nodes, i.e., control edges specify precedence relations between nodes. Further, the notion of *activity set* refers to any subset of the activities of a process schema, whereby the elements of such set do not need to meet structural requirements (e.g., compoundness). In turn, when referring to specific regions of a process schema, we use the notion of *process fragment*. Such a fragment refers to a sub-graph of a process schema with single entry and single exit node.

In addition to the described control flow elements, a process schema contains process-relevant *data objects* as well as *data edges* linking an activities with data objects. More precisely, a data edge either represents a read or write access of the respective activity to the referred data object.

At run-time, *process instances* are created and executed according to the predefined process schema. *Activity instances*, in turn, represent executions of single process steps (i.e., activities) of a particular process instance. Moreover, during run-time a particular data element may be written multiple times, resulting in a *series of data values* for it. Finally, the notion of *process instance set* refers to a set of process instances executed by the PAIS. In turn, the process instances of such a set may be enacted based on the same process schema, but may also run on different process schemes.

During the execution of a process instance, *events* may be triggered, either by the process instance itself (e.g., start / end event of the process instance), by a process node (e.g., start / end event of an activity instance, cf. Fig. 1), or by an external source (e.g., receipt of a message). We use the notion of *event* as general term for something happening during process execution. We assume that activity instances are executed according to the activity life cycle depicted in Fig. 1. When a process instance is started, its activities have state *Not Activated*. As soon as a particular activity becomes enabled, its enters state *Activated*. When a user starts the activity instance in this state, the latter switches to state *Started* and a corresponding *start event* is generated. As soon as the user completes the activity instance, its state switches to *Completed* and an *end event* is generated. Finally, non-executed activities enter state *Skipped* (e.g., activities contained in an outgoing path of an XOR-split not selected during run-time).

2.2 Time Patterns

Time Patterns (TP) represent temporal constraints commonly occurring in PAISs. In [18, 19], we identified 10 time patterns (cf. Fig. 2) and described them

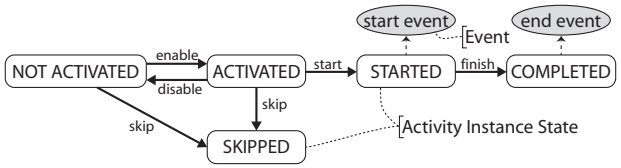

Fig. 1. Activity life cycle and relevant events

informally. We further provided empirical evidence for the relevance of these patterns, i.e., each time pattern was observed in at least 3 different application domains.

The 10 time patterns can be divided into 4 distinct categories based on their semantics (cf. Fig. 2). *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. Further, it 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 entire processes (e.g., activity deadlines). In turn, *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).

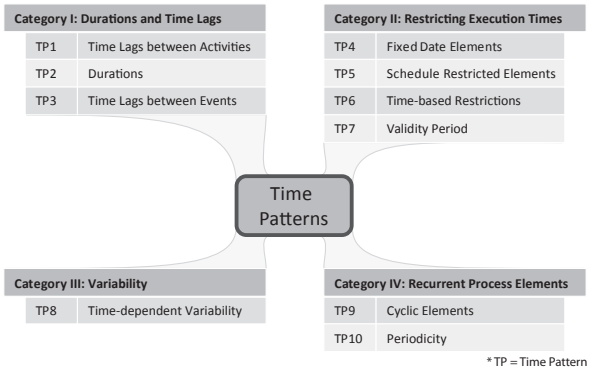

Fig. 2. Pattern catalogue [19]

Example 1 introduces a simple process along which we will illustrate selected time patterns (cf. Example 2). A basic understanding of these patterns is required in order to understand their formal definition given in Section 4.

Example 1 (Patient Treatment Process). *Consider the simplified patient treatment process depicted in Fig. 3. First, a doctor orders a medical procedure for his patient. Then, the responsible nurse makes an appointment with the department (e.g., radiology department) the procedure will take place. Before the treatment may be performed, 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.*

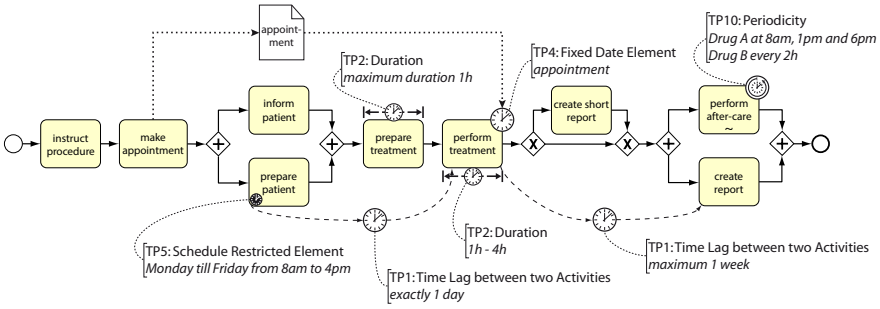

Fig. 3. Treatment process with temporal constraints

When considering the temporal perspective of this rather simple business process, a number of temporal constraints can be observed. Example 2 relates these temporal constraints to the time patterns from Fig. 2.

Example 2 (Temporal Constraints). *Consider again Fig. 3. The appointment of the* `treatment`*, made in the context of activity* `make appointment`*, must be observed during process execution. This constitutes a* ***Fixed Date Element*** *(TP4) restricting the earliest start date of the respective activity. Furthermore, the respective date will be set during run-time by activity* `make appointment` *and stored in data object* `appointment`*. In particular, this constraint affects the scheduling of preceding activities as well; e.g., the patient needs to be prepared exactly 1 day before his or her* `treatment` *takes place. This represents a* ***Time Lag between two Activities*** *(TP1) from the start of* `prepare patient` *to the start of* `perform treatment`*. Hence,* `preparation` *needs to be scheduled in accordance with the appointment of the* `treatment`*. Additionally,* `preparation` *of the patient may be only done 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 the ones of the* `treatment` *itself. Next, due to a tight schedule of the treatment room, the* `preparation` *of the treatment must not take more than 1 hour; otherwise the* `treatment` *is delayed. This represents a maximum* ***Duration*** *(TP2) on activity* `prepare treatment`*. The* `treatment`

itself, in turn, may take from 1 hour up to 4 hours, which represents a minimum and maximum ***Duration****, 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`*.*

This example demonstrates the diversity and complexity of the temporal perspective of business processes. It further illustrates why process modeling languages like BPMN are by far not sufficient to properly describe the temporal perspective of business processes. For example, time patterns like Schedule Restricted Elements (TP5), Time Lags between two arbitrary (not succeeding) Activities (TP1), and Periodicity (TP10) cannot be properly described using BPMN [19]. Similarly, Example 2 emphasizes the need for a precise and formal semantics of the different time patterns.

To cope with the variability and to keep the number of distinct patterns manageable, *design choices* (DC) allow for the parametrization of the time patterns [18, 19]. 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 *pattern variant*. Usually, existing PAISs only support a subset of the design choices available for a particular pattern (i.e., pattern variants). Note that design choices influence the formal semantics of the time patterns as well. For example, consider the *Duration* pattern (TP2) applied to an activity. To be able to exactly decide on the semantics of this pattern, it becomes necessary to specify whether the duration corresponds to a minimum value, a maximum value, or both (i.e., a time interval); i.e., the semantics of this time pattern is determined by taking the respective design choice into account. Hence, the formalizations provided in this paper consider design choices as well. Finally, when applying a specific pattern variant to a process schema the particular restriction represented by the pattern occurrences needs to be determined (e.g., the time of the appointment for pattern *Fixed Date Element*, or the duration value for pattern *Duration*). For this purpose *parameter values* allow determining the specific temporal constraint represented by a pattern occurrence. Parameter values, in turn, may either be set at build-time (i.e., all process instances share the same value) or at run-time, i.e., when creating or executing process instances. In the latter case, the particular parameter value is stored in a data object of the process instance when creating it or executing an activity. The stored data value is then read by the corresponding pattern occurrence during run-time. For this purpose, a pattern occurrence may be linked to a data object by means of a data edge (e.g., data object `appointment` in Fig. 3).

3 Research Method

This paper complements our previous work on workflow time patterns. Our goal is to provide a formal semantics for each of the time patterns independent of a particular process meta model. The overall procedure comprising pattern identification and the definition of pattern semantics is illustrated in Fig. 4. This section describes the research method (cf. Fig. 4) applied when identifying the time patterns (i.e., selection criteria and identification procedure) [18, 19]. We further describe essential requirements for defining pattern semantics and the procedure applied for their elicitation. Finally, we sketch the validation procedure we applied to ensure that pattern semantics meets the one expected by domain experts (cf. Fig. 4).

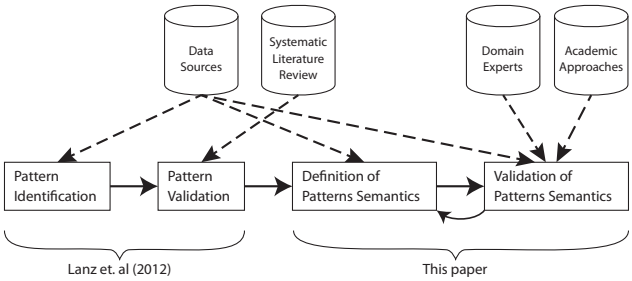

Fig. 4. Applied research method

3.1 Initial Pattern Identification

Selection criteria for the time patterns have been "patterns covering temporal aspects relevant for the modeling and control of business processes and activities respectively" [19]. In particular, other 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 as well.

To ground the time patterns on a solid basis, first of all, a candidate list was created based on the knowledge and experiences of the authors in this field. *Design choices* were introduced for parameterizing the patterns in order to keep the number of different patterns manageable. Following this, large sets of cases and process schemata were analyzed (cf. Table 1) in order to provide empirical evidence for each of the time patterns and to further refine the pattern list. This finally resulted in a set of 10 time patterns.

To validate the time patterns and to further asses their relevance and completeness, in addition, a systematic literature review was conducted [19]. Finally, different approaches and tools were evaluated regarding the support of the time patterns proposed [19].

3.2 Pattern Semantics: Requirements and Procedure

On the one hand, formal pattern semantics should be as generic as possible, on the other, it must be as specific as necessary to avoid ambiguities. Additionally, it must consider all *pattern variants* that may be configured due to different combinations of design choices.

We first analyze the data sources used for pattern identification (cf. Table 1) to elaborate on the intended semantics of each pattern occurrence. Next, we combine semantics of all occurrences of a specific pattern to obtain its overall semantics. Thereby, we specifically consider the respective variant of the pattern occurrence. Finally, if possible, we remove unnecessary restrictions from the overall patterns semantics.

Data Sources	
Healthcare Domain (Women Hospital) [26, 27, 28]	88 processes
Healthcare Domain (Internal Medicine) [29, 30, 31, 32, 33]	93 processes
Automotive Domain [34, 35]	55 processes
On-demand Air Service [36, 37]	2 processes
Airline Catering Services [38]	1 process
Transportation [39]	9 processes
Software Engineering [40]	10 processes
Event Marketing	1 process
Financial Service	10 processes
Municipality Processes [41]	4 processes
	Total: 273 processes

Table 1. Data sources [19]

As emphasized, any pattern semantics is only useful for PAIS engineers if it is defined independent of a particular process modeling language (e.g., BPMN, EPC) or process modeling paradigm (e.g., imperative vs. declarative). Workflow patterns have been defined based on languages such as Petri Nets [10] or pi-calculus [42], which have an inherent formal semantics. However, such formalisms cannot be used to define the semantics of time patterns without significant extensions since they do not consider the temporal perspective at all. Beyond that, different techniques commonly used for describing the temporal properties of a system exist. Examples include temporal logics [43], timed petri nets [44], and timed automata [45]. In principle, these techniques can be also used for describing the formal semantics of time patterns as well. However, most of them are either related to a particular process modeling paradigm or cannot be intuitively applied to describe high-level temporal constraints.

To be independent of such particulars, this paper uses *execution traces* as basis of pattern formalization due to their language-independence as well as their close relation to the execution semantics of processes. An execution trace represents a possible execution of a process schema indicating all events that occurred during the execution of the respective process instance together with their time stamps (cf. Sec. 4.1 for details). Based on execution traces 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 are independent of the particular process modeling language and paradigm; yet we are still able to precisely define pattern semantics.

3.3 Pattern Semantics: Validation

To validate the formal semantics of the time patterns, we reconsider each pattern occurrence observed in any data source (cf. Table 1), and compare the described semantics with our formal definitions. If the semantics of one of the pattern occurrences in our data sources is not precise enough, we ask domain experts to provide further details. Whenever necessary, we discuss a pattern occurrence with the respective domain expert in order to be sure that our pattern semantics meets the one the expert has in mind.

To further validate the formal semantics, we present it to different domain experts and process engineers and ask them whether they meet the description of the respective time pattern as well a their own expectations. For this purpose, we develop a representative set of interactive animations of the different time patterns and respective pattern variants displaying the defined formal semantics (a sub-set of these animations can be found on the project website [20]). We then present these animations to different subjects (i.e., domain experts and process engineers) and ask them to (a) describe the meaning of the shown pattern variant and (b) whether or not this matches their expectation. Regarding process engineers, we further show them a set of process fragments each exhibiting a set of temporal constraints. We then ask them to describe their interpretation of the respective process fragment. Next, for half of the process fragments we ask the subject to give a set of execution traces of the process fragment which are either consistent or inconsistent with respective temporal constraints. For the other half of cases we present the subject a set of corresponding execution traces and ask them whether or not in their opinion they are consistent with the temporal constraints of the process fragment.

In case a modification of the formal definition of the semantics of a particular time pattern is required, we implement it. Then the overall validation process is started from scratch for that particular pattern semantics. This is to ensure that the respective modification has no unintended consequences.

In a final step, we compare our formal semantics with existing approaches for verifying the consistency of the temporal perspective (cf. Sec. 5). Since these approaches are based on some formalism used for verification (e.g., temporal constraint networks [46], project network technique [47]) they come along with an inherent semantics that can be derived from the respective formalism. Thus, for each pattern supported by the respective approach we compare our formal semantics with the one of the respective approach. The latter, in turn, we obtain by combining the formalism used for verification and the translation procedure from the process schema to this formalism.

4 On the Formal Semantics of Time Patterns

Since we use execution traces to define pattern semantics, Section 4.1 first defines basic notions (e.g., *event* and *trace*) used throughout this paper. Section 4.2 then provides the formal semantics of each of the time patterns presented in [19].

4.1 Temporal Execution Traces

We use the notion of *event* as general term for something that happens during the execution of a process instance (cf. Fig. 5). For example, the start and end of an activity, executed in the context of a particular process instance, constitute such events (cf. Fig. 1 and Fig. 5). However, external events (e.g., reaching a milestone or receiving a message) may exist as well. Additionally, there are intermediate activity events, which may be triggered inside a long-running activity (cf. Fig. 5).

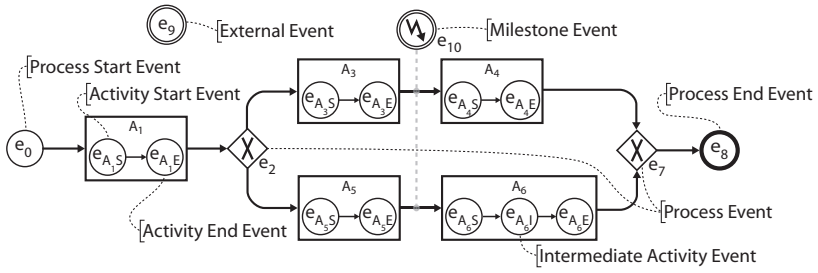

Fig. 5. Events occurring during a process

Definition 1 (Process schema, Process instance, Event, Event occurrence). *Let $\mathcal{PS}$ be the set of all process schemes and let $\mathcal{E}$ 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). We assume that all events related to the

execution of a particular process instance are recorded in a temporal execution trace together with a time stamp designating their time of occurrence.[1] Formally:

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

Since pattern semantics is defined based on events and event occurrences we provide a definition of nodes and activities based on events. In particular, respective definitions are based on the events that may occur during the execution of the particular node (activity). Thereby, nodes represent both activities and control connectors (cf. Sec. 2.1).

Definition 3 (Node and Activity). *Let $\mathcal{PS}$ be the set of all process schemes and $\mathcal{N}_S \subset 2^{\mathcal{E}_S}$ be the set of all nodes based on which process schema $S \in \mathcal{PS}$ is specified. Further, node $n \in \mathcal{N}_S$ is defined as unique, non-empty set of events $n \subset \mathcal{E}_S$ (i.e., $\forall n \in \mathcal{N}_S : n \neq \varnothing$) Thereby, the sets of events of all nodes of a process schema S are disjoint: $\forall n, m \in \mathcal{N}_S, n \neq m : n \cap m = \varnothing$. An occurrence of a node n in a temporal trace σ_S is marked by the occurrence of at least one of the respective events, i.e., $\exists e \in n, \exists t \in \mathcal{C} : \varphi_e = (e, t) \in \sigma_S$.*

An activity, in turn, is a special kind of node with a single start and a single end event (cf. Sec. 2.1). Let $\mathcal{A}_S \subseteq \mathcal{N}_S$ be the total set of activities based on which process schema $S \in \mathcal{PS}$ is specified. 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., $\forall a \in \mathcal{A}_S : \exists e_{a_S}, e_{a_E} \in \mathcal{E}_S : \{e_{a_S}, e_{a_E}\} \subseteq a$. In turn, 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., $\exists t_1, t_2 \in \mathcal{C} : \varphi_{a_S} = (e_{a_S}, t_1) \in \sigma_S \wedge \varphi_{a_E} = (e_{a_E}, t_2) \in \sigma_S$. ◊

Without loss of generality, we can assume that the processing of control connectors (e.g., XOR-splits, and AND-joins) does not take any time during process execution. If a control connector does consume time during its execution (e.g., evaluating an XOR decision), this can be represented by an activity directly preceding the control connector.

To illustrate these definitions consider Example 3.

[1] A temporal execution trace can usually be extracted from the *execution log* generated by a PAIS during the execution of a process instance [1].

Example 3 (Temporal Execution Trace). *Consider Fig. 5. A temporal execution trace of the depicted process schema may be as follows:*[2]

$$\sigma = \langle (e_0, 2), (e_{A_1 S}, 5), (e_{A_1 E}, 10), (e_2, 11), (e_{A_5 S}, 15), (e_{A_5 E}, 18), \\ (e_{10}, 19), (e_{A_6 S}, 30), (e_{A_6 I}, 32), (e_{A_6 E}, 37), (e_7, 38), (e_8, 39) \rangle$$

This trace indicates that the particular process instance was started (i.e., e_0) at time point 2. At time point 5, activity A_1 was started ($e_{A_1 S}$). In turn, A_1 was completed at time point 10 ($e_{A_1 E}$). Next, event e_2 occurred at time point 11 and the lower path was chosen, which is indicated by the occurrence of start event $e_{A_5 S}$ related to activity A_5. Further note that none of the events of the upper path occur within the trace as the respective path has been deselected. After completing activity A_5 ($e_{A_5 E}$), milestone event e_{10} was triggered. Next, A_6 was started at time point 30. Furthermore, during the execution of A_6, the intermediate activity event $e_{A_6 I}$ was triggered. Finally, after completion of A_6, events e_7 and e_8 were triggered completing the process instance.

A path constitutes a connected part of a process schema with a single entry and a single exit node.

Definition 4 (Path). *The set of all paths within a process schema $S \in \mathcal{PS}$ is denoted as $\mathcal{P}_S$. In turn, a path $\varrho \in \mathcal{P}_S$ is represented as subset $E_\varrho \subseteq \mathcal{E}_S$ of all events producible on process schema S, where E_ϱ contains all events that may occur during the execution of path ϱ. In particular, a path is compound and has a single entry and a single exit node connecting it with the remaining process. Thereby, for each split node, the respective join node is also contained and vice versa.* ◊

Paths are also denoted as single-entry single-exit regions (SESE) [48].

Example 4 (Paths). *Consider Fig. 5 which shows a process schema with several different paths. For example, activities A_3 and A_4 (i.e., events $e_{A_3 S}$, $e_{A_3 E}$, $e_{A_4 S}$, and $e_{A_4 E}$) constitute a path. Likewise, activities A_5 and A_6 represent a path. Furthermore, the process fragment containing activities A_1, A_3, A_4, A_5, and A_6 as well as nodes e_2 and e_7 constitutes another path. Finally, note that according to Definition 4 each of the activities represents a path by its own.*

When defining time pattern semantics, we must also deal with patterns for which the iteration of a loop structure needs to be taken into account.

Example 5 (Cyclic Elements). *Consider the process depicted in the upper part of Fig. 6. It contains two nested loops: the inner loop L_2 contains two alternative branches either executing activity A_6 or A_7. Therefore, 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*

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

one to which the source activity belongs. Assume that the cyclic element only restricts the maximum time lag between directly succeeding iterations of the two activities (design choice K[a]), i.e., only if A_7 is executed in an iteration of Loop L_2 directly after A_6, but without executing another instance of A_6, A_7 or A_{10} in between, the respective maximum time lag must be obeyed. To formalize this semantics, it is not sufficient to introduce an iteration counter for activities. For instance, considering Example 5 it cannot always be detected whether another iteration of loop L_1 has been executed between the two instances of activities A_6 and A_7. Hence, the current iteration of the outer loop L_1 must be considered as well.

Generally, it is necessary to always consider the iteration of the inner-most loop in respect to the iteration of any surrounding loop. When considering Fig. 6, 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. 6). Based on this, we define the notion of (loop) iteration as follows:

Definition 5 (Iteration). *Let $S \in \mathcal{PS}$ be a process schema. Then: a loop L is a specific path $\varrho \in \mathcal{P}_S$ with the loop start node as entry node and the corresponding loop end node as exit node. The set of possible iterations of process schema $S \in \mathcal{PS}$ is then given as $\mathcal{I}_S \subseteq 2^{\mathcal{P}_S \times \mathbb{N}}$. Based on $\mathcal{I}_S$, 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 respective process schema and L_i $(1 \leqslant i \leqslant k)$ the i-th loop structure. In turn, n_{L_i} $(1 \leqslant i \leqslant k)$ designates the iteration count of loop L_i with respect to its directly surrounding loop L_{i-1}.

In turn, function $\mathrm{iter}(\sigma_S, \varphi)$ *returns the current iteration of the inner-most loop surrounding event φ^e. Formally:*

$$\mathrm{iter} : \mathcal{Q}_S \times \Phi_S \mapsto \mathcal{I}_S$$

with

$$\mathrm{iter}(\sigma_S, \varphi) = \langle (L_0 : n_{L_0}), \ldots, (L_k : n_{L_k}) \rangle$$

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

Example 6 (Iteration). *In Fig. 6 below the depicted process schema, a temporal execution trace, together with the respective value of* $\mathrm{iter}(\sigma_S, \varphi)$ *for each of the events in the trace, is shown. For example, regarding the occurrence of event $e_{A_7 S}$ during the 2nd iteration of loop L_2 within the 1st iteration of loop L_1, we obtain*

$$\mathrm{iter}\,(\sigma_S, (e_{A_7 S}, \cdot)) = \langle (L_0 : 1), (L_1 : 1), (L_2 : 2) \rangle$$

This is indicated in the second line of the execution trace depicted in Fig. 6.

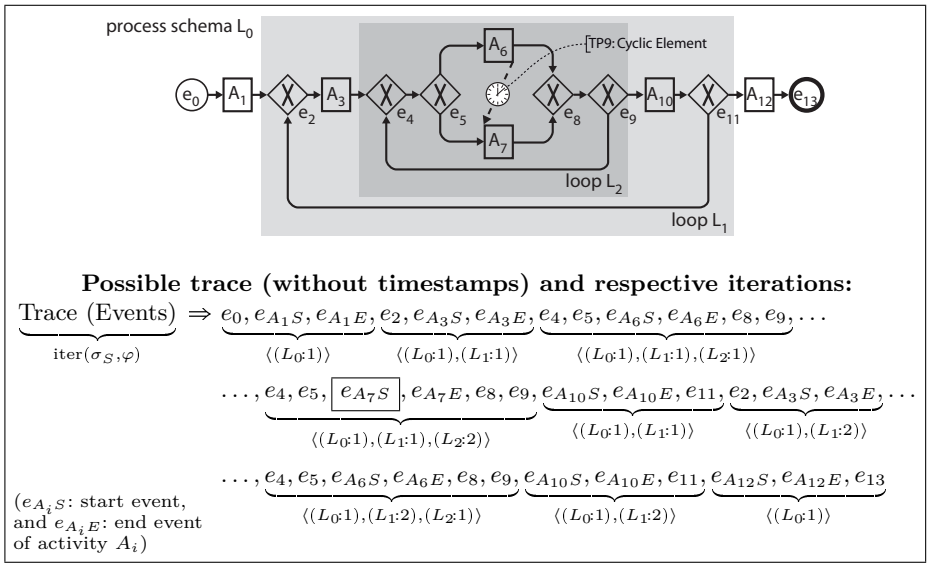

Fig. 6. Nested loops and iterations

Based on Definition 5, Table 2 summarizes useful notions. Note that, these facilitate the formalization of the time patterns.

Since a particular event may occur multiple times within a trace (e.g., in connection with loops), we further define function $\text{occur}(\sigma_S, e)$. It returns all event occurrences of an event e within a trace :

Definition 6 (Trace Occurrences). *Let $S \in \mathcal{PS}$ be a process schema and σ_S a temporal trace on S. Then:* $\text{occur}(\sigma_S, e)$ *corresponds to a function returning all occurrences $\varphi = (e, \cdot)$ of event $e \in \mathcal{E}_S$ within σ_S. Formally:*

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

with

$$\text{occur}(\sigma_S, e) = \{\varphi \in \sigma_S | \exists t \in \mathcal{C} : \varphi = (e, t) \in \sigma_S\}$$

◊

Note that Definition 6 implies that $\text{occur}(\sigma_S, e) = \varnothing$ holds, if $\nexists t \in \mathcal{C} : (e, t) \in \sigma_S$, e.g., in case the respective path has been skipped (cf. Example 3).

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. Definition 3). Therefore, we define function $\text{occur}(\sigma_S, a)$, which returns all instances of a particular activity a (i.e., occurrences of the start and end events) within a trace:

Definition 7 (Activity Occurrence). *Let $S \in \mathcal{PS}$ be a process schema and σ_S a temporal trace on S. Then:* $\text{occur}(\sigma_S, a)$ *is a function that 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 $a \in \mathcal{A}_S$*

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

$$\exists \varphi \in \sigma_S : \text{iter}(\sigma_S, \varphi) = I$$

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

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

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 $n \geqslant 1$:

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

or

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

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

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

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

Table 2. Useful notions

within σ_S.[3] *Thereby, two occurrences of events e_{a_S} and e_{a_E} belong to the same activity instances if they occurred within the same iteration. Formally:*

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

with

$$\text{occur}(\sigma_S, a) = \{(\varphi_S, \varphi_E) \in \Phi_S \times \Phi_S \mid \varphi_S \in \text{occur}(\sigma_S, e_{a_S}) \wedge \varphi_E \in \text{occur}(\sigma_S, e_{a_E}) \wedge \text{iter}(\sigma_S, \varphi_S) = \text{iter}(\sigma_S, \varphi_E)\}$$

$\Diamond$

Finally, for the sake of readability, we will omit subscript S in the formulas if the respective process schema is evident in the given context.

[3] Remember that $\{e_{a_S}, e_{a_E}\} \subseteq a$ (cf. Definition 3)

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

Note that the notions from Definitions 1-8 are independent of any particular process meta model. Based on them, we describe the formal semantics of the time patterns. Thereby, we do not differentiate whether the parameter values of a pattern are set during build-time, process creation-time, or run-time (*Design Choice A*) as this does not affect formal pattern semantics. Nevertheless, we consider which value of a parameter is valid for a specific 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 value is relevant for a particular instance of a pattern occurrence when defining pattern semantics.

Example 7 (Run-time Parameter Values). *Consider the process schema depicted in Fig. 7. The process contains a maximum time lag between activities A_2 and A_4. The respective value of the time lag is provided during run-time by data object* `time lag`*. Data object* `time lag`*, in turn, is written by activities A_1 and A_5. It is important to note that activity A_5 may be executed (or more important be completed) either before, during, or after executing activity A_4. Depending on the completion time of activity A_5 the time lag may use different parameter values, i.e., values of data object* `time lag`*. It is therefore important to know which of these values is valid for a particular instance of the time lag.*

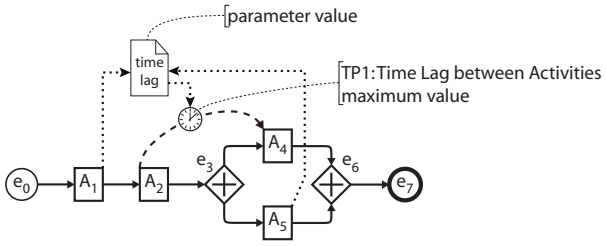

Fig. 7. Run-time parameter value and concurrent change

Finally, to structure formalization effects, we bundle time patterns having similar or related foundations (and therefore formalization).

Our first set of time patterns having 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. What kind of relative time distance is expressed, constitutes a pattern-specific design choice (*Design Choice D*); i.e., the time distance can be a minimum value, a maximum value, or an interval (i.e., both minimum and maximum). The kind of events to which a specific pattern can be applied may be also restricted by the pattern itself; e.g., a time lag between activities may only be applied to the start or end events of two different activities. Furthermore, only pairs of event occurrences whose iterations are compliant with the respective pattern may be considered; e.g., for time pattern activity *Duration* (TP2; *DC C[a]*), the occurrences of the start and end event of the respective activity must belong to the same iteration.

Definition 9 (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 source and target events of the time distance, i.e., the events between which the time distance is specified. 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 the target event ψ. Thereby, a pair of event occurrences is valid if their iterations are compliant with the respective pattern. This is expressed by pattern-specific function $\text{valid} : \mathcal{Q} \times \Phi \times \Phi \rightarrow \mathit{Boolean}$. *In turn, function* $\text{distance} : \mathcal{Q} \times \mathcal{E} \times \mathcal{E} \times \mathcal{C} \rightarrow 2^{\mathcal{D}}$ *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, maximum distance, or time interval; cf. Design Choice D). In turn, this is defined by function*

$$\text{compareR} : \mathcal{C} \times \mathcal{C} \times 2^{\mathcal{D}} \rightarrow \mathit{Boolean}$$

with

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

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

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

◊

Formula (1) expresses that 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*). In this context, note that only the parameter value (i.e., the data value) which is valid at the occurrence time of the target event is considered (see distance$(\cdot,\cdot,\cdot,\psi^t)$ and cf. Example 7). Based on this general definition, the semantics of time patterns TP1, TP2, TP3, and TP9 will be defined.

Time pattern TP1 restricts the time lag between two activities A and B.

Pattern Semantics 1 (TP1: Time Lags between Activities). The semantics of time pattern TP1 can be defined using Formula (1) by applying the following restrictions to e_X, e_Y, and valid(σ,φ,ψ):

(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.[4] Consequently, it holds:

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

Restriction (a) expresses, for example, that for an End-Start relation between two activities (e.g., A_3 and A_6 in Fig. 8), 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.

In turn, restriction (b) expresses that the two activities (i.e, respective event occurrences) must either belong to the same or to an adjacent iteration. The second part of (b) can then be interpreted as follows: The definition of the pattern semantics needs to be valid in connection with loops as well. As illustrated in Fig. 8, considering time lags between two activities (TP1) of which one resides inside a loop and the other one outside this loop (e.g., a time lag between activities A_1 and A_3 in Fig. 8) is a challenging task due to the unclear semantics of such scenario. This becomes even more complicated when considering nested loops as well. For example, consider a time lag between activities A_1 and A_6 in Fig. 8. There exist several possible semantics for such a time lag. First, time lags whose source activity is contained in one loop and whose target activity is inside a nested loop (cf. Fig. 8) might only apply to the first or last iteration of the nested loop. Second, such "inbound" time lag may be applied to all iterations of the inner loop as well (effectively leading to different semantics for minimum and

[4] Note that the adjacency operator $\parallel$ has been defined in Table 2

maximum time lags). Third, the time lag may be augmented by meta information describing the iteration or iterations it applies to. Note that the same holds for time lags whose source activity is inside a nested loop and whose target activity is outside that loop (cf. Fig. 8). To resolve this issue, the current definition of the semantics of pattern TP1 restricts the application of such a time lag to adjacent iterations of respective loops, i.e., it restricts an "inbound" time lag to the first iteration of a loop and an "outbound" time lag to the last iteration of a loop. If necessary, this can be extended to specific iterations of the respective loops.

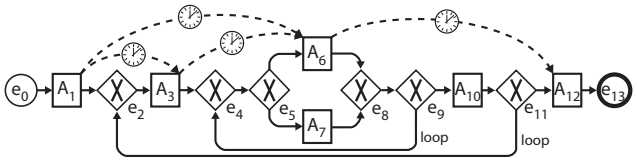

Fig. 8. Time lags and loops

Example 8 (Pattern Semantics 1). *We illustrate Pattern Semantics 1 for the process schema depicted in Fig. 9. This schema contains a minimum time lag of 10 between the end of activity A_1 and the start of activity A_6, and a maximum time lag of 7 between the start of A_3 and the start of A_7. Based on this process schema, for example, traces σ_1-σ_3 may be produced.*

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

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

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

Trace σ_1 is compliant with the set of temporal constraints defined on the process schema. For example, the time lag between A_1 and A_6 is observed as

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

and the time lag between A_3 and A_7 as

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

In turn, trace σ_2 is not compliant with the temporal constraints defined by the process schema. Although the time lag between A_3 and A_7 is fulfilled, as activity A_3 is not executed, the one between A_1 and A_6 is not fulfilled:

$$\varphi^t_{e_{A_1E}} + \min(\text{distance}(\sigma, e_{A_1S}, e_{A_6E}, \varphi^t_{e_{A_6E}})) = 5 + 10 = 15 \nleqslant \varphi^t_{e_{A_6S}} = 13.$$

Finally, trace σ_3 is not compliant with the process schema. This time the time lag between A_1 and A_6 is observed, but the time lag between A_3 and A_7 is not, because it holds

$$\varphi^t_{e_{A_7S}} = 23 \nleqslant \varphi^t_{e_{A_3S}} + \max(\text{distance}(\sigma, e_{A_3S}, e_{A_7S}, \varphi^t_{e_{A_7S}})) = 10 + 7 = 17.$$

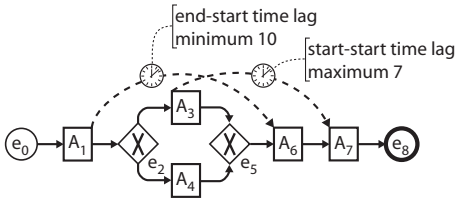

Fig. 9. Illustration of Pattern Semantics 1

We next consider the semantics of time pattern TP3. Remember that TP3 restricts the time lag between two arbitrary events.

Pattern Semantics 2 (TP3: Time Lags between Arbitrary Events). The semantics of TP3 can be expressed based on Formula (1) and applying the following restrictions to e_X, e_Y, and $\text{valid}(\sigma, \varphi, \psi)$:

(a) Both e_X and e_Y constitute arbitrary events.
(b) As for Pattern Semantics 1, respective events have to belong either to the same or an adjacent iteration. Hence, it holds:[5]

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

As can be seen in the context of Pattern Semantics 2, TP3 constitutes a generalization of TP1 [19], effectively removing restriction (a) as specified for Pattern Semantics 1.

Time pattern TP9 restricts the time lag between activities A and B across iterations.[6] As discussed in [19], time pattern TP9 is a variant of time pattern TP1. However, instead of enforcing the two activity instances to be inside 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 to which the source activity belongs.

[5] For the sake of simplicity we assume same restrictions as for TP1. We are aware that this might not always be desired. However, it is no problem to adapt this definition for cases where other relations between iterations are required.
[6] 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) and by applying the following restrictions to e_X, e_Y, and valid(σ, φ, ψ):

(a) For e_X and e_Y, the same restrictions as for Pattern Semantics 1 apply.
(b) The activities and corresponding events of a cyclic element must either belong to directly succeeding iterations or to arbitrary, but succeeding iterations (*Design Choice K*). Hence, it holds:

$$\text{valid}(\sigma, \varphi, \psi) \equiv \begin{cases} \text{succ}(\text{iter}(\sigma, \varphi)) = \text{iter}(\sigma, \psi) & \text{if directly succeeding iterations (DC K[a])} \\ (\text{iter}(\sigma, \varphi) < \text{iter}(\sigma, \psi)) \wedge & \text{if arbitrary, but succeeding iterations (DC K[b])} \\ \quad (\nexists \omega \in \text{occur}(\sigma, \varphi^e) \cup \text{occur}(\sigma, \psi^e) : & \\ \quad\quad \text{iter}(\sigma, \varphi) < \text{iter}(\sigma, \omega) < \text{iter}(\sigma, \psi)) & \end{cases}$$

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 (*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(σ, φ))) corresponds 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(σ, ψ)). However, there may not be any occurrence of either the source φ^e or the target ψ^e event inbetween these two iterations.

Time pattern TP2 allows expressing that a particular activity, activity set, process, or set of process instances must obey a duration restriction. Its semantics is quite similar to Pattern Semantics 1-3 since it considers the relative time distance between events as well.

Pattern Semantics 4 (TP2: Durations). For an activity or process (*Design Choice C[a]* and *C[c]*), respectively, the semantics of TP2 is defined based on Formula (1) by applying the following restrictions to e_X, e_Y, and valid(σ, φ, ψ)

(a) Depending on the chosen granularity (i.e., activity or process) to which the pattern is applied 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 therefore the same iteration, i.e,

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

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

– For a minimum duration (*Design Choice D[a]*), it must hold that 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 which satisfies the respective minimum duration:

$$\forall a \in A : \forall(\varphi_S, \varphi_E) \in \mathit{occur}(\tau, a) \Rightarrow \left[\begin{array}{l} \exists e_X \in E_X, \exists e_Y \in E_Y : \exists \mu \in \mathrm{occur}(\sigma, e_X), \exists \nu \in \mathrm{occur}(\sigma, e_Y) : \\ \quad \mathrm{valid}(\sigma, \mu, \nu) \wedge \mathrm{iter}(\sigma, \varphi_S) = \mathrm{iter}(\sigma, \mu) \\ \quad \Rightarrow \mu^t + \min(\mathrm{duration}(\sigma, A, \nu^t)) \leqslant \nu^t \end{array}\right] \tag{2}$$

Each iteration may have several valid event pairs. However, only one of these pairs must obey the given minimum duration. In this context, note that the duration of the activity set corresponds 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 $e_Y \in E_Y$

$$\begin{array}{l} \forall e_X \in E_X, \forall e_Y \in E_Y : \forall \varphi \in \mathrm{occur}(\sigma, e_X), \forall \psi \in \mathrm{occur}(\sigma, e_Y) : \\ \qquad \mathrm{valid}(\sigma, \varphi, \psi) \Rightarrow \psi^t \leqslant \varphi^t + \max(\mathrm{duration}(\sigma, A, \psi^t)) \end{array} \tag{3}$$

– For a time interval (*DC D[c]*), both Formula (2) and Formula (3) must be satisfied.

For handling a set of process instances (*DC C[b]*), 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 set. For the sake of brevity, 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 is never executed within the respective process instance.

As another set of time patterns with similar foundations we consider patterns 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 process (*Design Choice C*). Finally, they can restrict the earliest start, latest start, earliest completion, or latest completion date of the activity or process, respectively (*Design Choice F*).

Definition 10 (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* $\mathrm{date} : \mathcal{Q} \times \mathcal{E} \times \mathcal{C} \to \mathcal{C}$ *represents the parameter value of the pattern occurrence as it is effective at time t. Finally, whether 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) and is expressed by function

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

with

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

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

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

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 allows expressing that a particular activity (process) instance must be executed in relation to a particular date.

Pattern Semantics 5 (TP4: Fixed Date Element). The semantics of time pattern TP4 can be expressed using Formula (4). In this context, $\text{date}(\sigma, e, \varphi^t)$ corresponds to a function returning the parameter value of the pattern occurrence as it is effective for process instance σ at the time of event occurrence $\varphi = (e, t)$.

Example 9 (Pattern Semantics 5). *We illustrate Pattern Semantics 5 along the process from Fig. 10. It contains a fixed date constraining the latest end date of activity* A_2. *In turn, the parameter value of the fixed date (i.e., the value of function* date*) is determined by activity* A_1 *and may be modified by activity* A_3.

Traces σ_1 *and* σ_2 *may be produced based on this process schema.*

$$\sigma_1 = \Big\langle (e_0, 0), (e_{A_1S}, 2), (e_{A_1E}, 4)_{\{\text{date}(\sigma, e_{A_2E}, t \geqslant 4) \leftarrow 7\}}, (e_{A_2S}, 5), (e_{A_3S}, 6), (e_{A_2E}, 7), (e_{A_3E}, 10)_{\{\text{date}(\sigma, e_{A_2E}, t \geqslant 10) \leftarrow 12\}}, (e_4, 12) \Big\rangle$$

$$\sigma_2 = \Big\langle (e_0, 0), (e_{A_1S}, 2), (e_{A_1E}, 4)_{\{\text{date}(\sigma, e_{A_2E}, t \geqslant 4) \leftarrow 12\}}, (e_{A_3S}, 5), (e_{A_3E}, 6)_{\{\text{date}(\sigma, e_{A_2E}, t \geqslant 6) \leftarrow 8\}}, (e_{A_2S}, 7), (e_{A_2E}, 9), (e_4, 12) \Big\rangle$$

Thereby, $(e_{A_1E}, 4)_{\{\text{date}(\sigma, e_{A_2E}, t \geqslant 4) \leftarrow 7\}}$, *for example, indicates that after the occurrence of event* e_{A_1E} *the value of* $\text{date}(\sigma, e_{A_2E}, \cdot)$ *is changed to* 7, *i.e.,* $\forall t \geqslant 4 : \text{date}(\sigma, e_{A_2E}, t) = 7$ *(cf. trace* σ_1*).*

σ_1 complies with the fixed date constraining activity A_2 as the parameter value of this fixed date is set to 7 *by A_1 and not modified by A_3 before completing A_2. Hence, for the completion event of A_2 it holds*

$$\varphi^t_{A_2E} = 7 \quad \leqslant \quad \text{date}(\sigma, e_{A_2E}, \varphi^t_{A_2E}) = 7.$$

In turn, σ_2 is not compliant with the fixed date constraining A_2. Initially, the parameter value of this fixed date element is set to 12 *by activity A_1. However, before completing A_2, activity A_3 is executed. Further, A_3 changes the parameter value of the fixed date element to* 8*. Hence, for the completion of A_2 it holds*

$$\varphi^t_{A_2E} = 9 \quad \nleqslant \quad \text{date}(\sigma, e_{A_2E}, \varphi^t_{A_2E}) = 8.$$

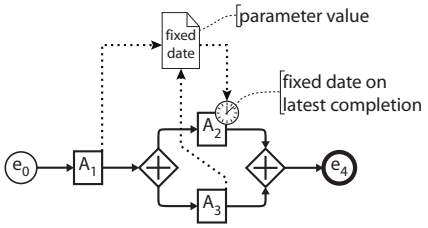

Fig. 10. Illustration of Pattern Semantics 5

Time pattern TP7 allows restricting the life time of an activity (or process) to a certain *validity period*.

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

Note that Pattern Semantics 5 and 6 are quite 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 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. Nevertheless, the impact the respective time patterns (i.e., TP4 and TP7) have on process execution is quite different; e.g., TP7 should be verified prior to creation of any process instance, whereas TP4 may only be verified during run-time. Consequently, semantics of time patterns TP4 an TP7 (i.e., Pattern Semantics 5 and 6) are defined separately.

We now consider the semantics of time pattern TP5 (*Schedule Restricted Elements*). As discussed in [19], 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. Since we do not want to restrict the representation of such a schedule (e.g., [49, 50]), we require that it can be materialized as a set of subsets of the absolute time points $\mathcal{C}$ (i.e., as a set of time slot).

Definition 11 (Schedule). *A schedule s is a possibly infinite set of continuous subsets (i.e., intervals / time slots) of the absolute time points $\mathcal{C}$. Formally:*

$$s \subset \{[t_{min}, t_{max}] | [t_{min}, t_{max}] \subseteq \mathcal{C}\} \subset 2^{\mathcal{C}}$$

$\Diamond$

Based on Definition 11, the semantics of time pattern TP5 (*Schedule Restricted Element*) can be defined:

Pattern Semantics 7 (TP5: Schedule Restricted Elements). Trace σ is 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 schedule : $\mathcal{Q} \times \mathcal{E} \times \mathcal{C} \rightarrow 2^{\mathcal{C}}$ represents the parameter value (i.e., the schedule) of the pattern occurrence as it is effective at time t. Formally:

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

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. 11 illustrate Pattern Semantics 7. It shows a simple process with a schedule restricting the start of activity A_1. Thereby, the schedule is given by expression $s = \{[5 + 10 * i, 9 + 10 * i] | i \geqslant 0\} = \{[5, 9], [15, 19], [25, 29], \ldots\}$.*

Traces σ_1 and σ_2 may be produced based on this process schema.

$$\sigma_1 = \langle (e_0, 0), (e_{A_1 S}, 2), (e_{A_1 E}, 9), (e_2, 10) \rangle$$

$$\sigma_2 = \langle (e_0, 0), (e_{A_1 S}, 6), (e_{A_1 E}, 11), (e_2, 12) \rangle$$

σ_1 is not compliant with the respective schedule restriction on A_1 as

$$\varphi^t_{A_1 S} = 2 \quad \notin \quad s = \{[5, 9], [15, 19], [25, 29], \ldots\}.$$

In turn, σ_2 is compliant with the schedule restriction:

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

Time-based Restrictions (TP6) allow us to restrict the number of executions of an activity, a set of activities, a process, or a set of process instances (*Design*

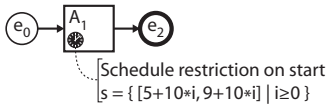

Fig. 11. Illustration of Pattern Semantics 7

Choice G) within a given time frame; e.g., a particular activity must not be executed more than once per day. This can be formalized by comparing the number of executions of the respective activities (processes) within the particular time frame on one side with a given minimum or maximum number of executions (*Design Choice H*) on the other.

When considering the semantics of pattern TP6, a subtle difference between the different pattern variants can be observed, which has not been covered by the original design choices yet [19]. To be able to completely define the semantics of time pattern TP6, *Design Choice U* must be considered as well (cf. Fig. 12). Based on this, the number of executions of a particular activity within a time frame can be defined.

Definition 12 (Executions per time frame). *Let σ_S be a temporal trace on process schema $S \in \mathcal{PS}$.*

Function compareT 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:

$$\text{compareT} : 2^C \times 2^C \rightarrow \mathit{Boolean}$$

with

$$\text{compareT}([t_{min}, t_{max}], [\varphi_S^t, \varphi_E^t]) \equiv \begin{cases} [t_{min}, t_{max}] \cap [\varphi_S^t, \varphi_E^t] \neq \varnothing & \textit{Overlapping time frames (DC U[a])} \\ [\varphi_S^t, \varphi_E^t] \subseteq [t_{min}, t_{max}] & \textit{Execution time frame contained in given time frame (DC U[b])} \\ \varphi_S^t \in [t_{min}, t_{max}] & \textit{Start of execution within given time frame (DC U[c])} \\ \varphi_E^t \in [t_{min}, t_{max}] & \textit{End of execution within given time frame (DC U[d])} \end{cases}$$

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:*

$$\text{executions} : \mathcal{Q}_S \times \mathcal{A}_S \times 2^{\mathcal{C}} \mapsto \mathbb{N}$$

with

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

◊

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

Fig. 12. Design Choice U for time pattern TP6

Based on Definition 12, the semantics of time pattern TP6 (*Time Based Restrictions*) can be defined:

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

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

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

For a minimum number of concurrent executions (*Design Choice H[a]*), the same formula applies, if using "$\geqslant n$" at the end.

– In case of 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 $s = \{[t_{min}, t_{max}] | [t_{min}, t_{max}] \subseteq C\}$ (cf. Def. 11). Then: A temporal trace σ is compliant with a maximum number n of executions (*Design Choice H[a]*) of activities $A \subseteq \mathcal{A}$ per element of schedule s iff:

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

For a minimum number of executions per time period (*Design Choice H[b]*), the same formula applies, if using "$\geqslant n$" at the end.

For a set of activities belonging to different process instances (*Design Choice G[b]*) or a set of process instances (*Design Choice G[c]*), these sums must be calculated considering the set of temporal traces. 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. The second part of Patterns 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 to a minimum number of executions.

Time pattern TP8 (*Time-dependent Variability*) allows varying the control flow depending on time aspects. Thus, it restricts the set of events a 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 3).[7]

Each instance of the pattern is triggered by a decision point represented by a specific event e_d (e.g., an XOR-split; cf. event e_2 in Fig. 9). To formalize time pattern TP8 and to abstract from a concrete system implementation we define function eval. It allows evaluating the condition at the time the decision event e_d occurs for each of the possible paths ϱ. Thereby, the current iteration I and the current process instance σ are considered.

Definition 13 (Evaluate). *Function* eval *evaluates the condition associated with a path $\varrho \in \mathcal{P}$ at time t the decision point e_d occurs. Formally:*

$$\text{eval} : \mathcal{Q} \times \mathcal{P} \times \mathcal{E} \times \mathcal{C} \mapsto \mathit{Boolean}$$

◊

The concrete implementation of function eval depends on the formalism used to describe the conditions for path selection. Depending on the outcome of function eval, the sub-traces of the respective path (i.e., the traces producible by this path) must or must not occur within the trace. Based on Definition 13, the semantics of TP8 is defined as follows:

Pattern Semantics 9 (TP8: Time-dependent Variability). A temporal trace σ is compliant with a time-dependent XOR or a time-dependent late binding (*Design Choice J[a]*) iff for all paths ϱ, affected by the time-dependent variability, the following condition is met:

$$\begin{aligned}
&\forall \mu \in \text{occur}(\sigma, e_d) : \\
&\quad [\text{eval}(\sigma, \varrho, e_d, \mu^t) = \mathit{true} \\
&\qquad \Rightarrow E_\varrho = \varnothing \vee \exists e \in E_\varrho : \exists \varphi \in \text{occur}(\sigma, e) : \text{iter}(\sigma, \varphi) = \text{iter}(\sigma, \mu)] \qquad \text{(i)} \\
&\wedge \\
&\quad [\text{eval}(\sigma, \varrho, e_d, \mu^t) = \mathit{false} \\
&\qquad \Rightarrow \forall e \in E_\varrho : \forall \varphi \in \text{occur}(\sigma, e) : \text{iter}(\sigma, \varphi) \neq \text{iter}(\sigma, \mu)] \qquad \text{(ii)}
\end{aligned}$$

[7] In case of late binding a path only consists of a single activity representing the service chosen in this path.

In turn, the semantics of a time-dependent deferred choice (*Design Choice J[b]*) can be defined based on *Patterns Semantics 1*, i.e., this semantics is the same as for the a time lag between two activities plus the semantics of the deferred choice workflow pattern [10]. For sake of brevity, the respective definition is omitted here.

Pattern Semantics 9 can be interpreted as follows: For each occurrence of the decision event and for each outgoing path ϱ, it is checked whether the condition is fulfilled. In case this condition evaluates to *true* either the path must be empty or at least one of its events must occur in the trace with the same iteration as the occurrence of the decision event (i). Otherwise, none of the events of the deselected path may occur within the same iteration as the occurrence of the decision event (ii).

As discussed in [19], a **Periodicity** (Time Pattern 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. 13 and its potential realization shown inside the activity. The semantics of pattern TP10 can be defined based on the one of the patterns used for realizing the periodicity during run-time. Therefore, no explicit semantics is provided for this pattern.

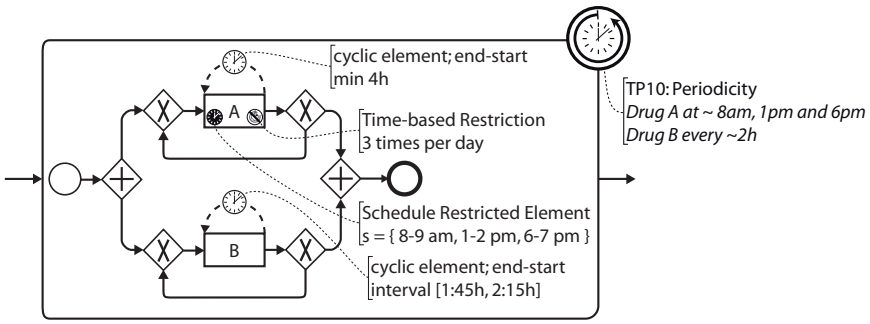

Fig. 13. Illustration of Periodicity and possible realization

4.3 Conclusion

Based on Pattern Semantics 1-9 we are able to check whether a given temporal execution trace is 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 all pattern variants. However, it must be ensured that for each possible path that may be selected during process execution, there exists at least one execution trace that is compliant with all

temporal constraints defined. 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 14 (Consistency). *A process schema $S \in \mathcal{PS}$ is consistent with the set of temporal constraints defined on S, if for each possible path $\varrho \in \mathcal{P}_S$ there exists at least one temporal execution trace $\sigma_S \in \mathcal{Q}_S$ that is compliant (cf. Definition 8) with the set of temporal constraints defined on S.* ◊

Example 11 (Consistency). *Consider the process depicted in Fig. 14. Now assume that at XOR-split e_3 the lower path is chosen. In this case a possible execution trace is*

$$\begin{aligned}\sigma_{\{e_3 \rightarrow lower\}} = \langle &(e_0, 0), (e_{A_1S}, 1), (e_{A_1E}, 4), (e_2, 5), (e_3, 6), (e_{A_8S}, 9), (e_{A_8E}, 15),\\ &(e_{A_9S}, 18), (e_{A_5S}, 20), (e_{A_9E}, 25), (e_{A_{10}S}, 28), (e_{A_5E}, 35), (e_6, 36),\\ &(e_{A_7S}, 37), (e_{A_{10}E}, 40), (e_{A_7E}, 45), (e_{11}, 46), (e_{A_{12}S}, 48),\\ &(e_{A_{12}E}, 58), (e_{13}, 59)\rangle\end{aligned}$$

This particular execution trace complies with all temporal constraints on the process schema. For example, maximum process duration is satisfied as $\varphi^t_{e_{13}} - \varphi^t_{e_0} = 59 \leqslant 60$. Moreover, all duration constraints are observed. Finally, the two time lags between A_1 and A_{10} as well as between A_9 and A_{12}, which are active for this trace, are satisfied as $\varphi^t_{e_{A_1E}} + 25 \geqslant \varphi^t_{e_{A_{10}S}}$ and $\varphi^t_{e_{A_9S}} + 35 \geqslant \varphi^t_{e_{A_{12}S}}$.

However, in case the upper path is chosen at XOR-split e_3, no valid execution traces exists that is compliant with all temporal constraint on the process schema. In particular, the path determining the shortest time to complete the process (i.e., the critical path) is given by $e_0, A_1, e_2, e_3, A_4, e_6, A_7, e_{11}, A_{12}$, and e_{13}; this sums up to a minimum process duration of 72. In turn, this violates the maximum process duration of 60. Consequently, the process schema is inconsistent, as there exists an execution path for which no valid trace can be found.

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 activity duration constraints defined for activities A_4 and A_{10} cannot be completely satisfied at the same time. The earliest possible time, activity A_4 may be completed after completing A_1 corresponds to $\varphi^t_{e_{A_1E}} + 20 + 30$. In turn, the latest possible time A_{10} may be completed after completing A_1 corresponds to $\varphi^t_{e_{A_1E}} + 25 + 10$. Hence, $(\varphi^t_{e_{A_1E}} + 25 + 10) - (\varphi^t_{e_{A_1E}} + 20 + 30) = 35 - 50 = -15 \notin [-10, 10]$ holds. Consequently, these constraints will never be completely satisfied at the same time.

Example 11 indicates the complexity for verifying the consistency of a process schema in respect to its temporal constraints [24, 25]. Things become even more complicated due to the fact that not all time patterns can be verified at build-time [19]. For example, the fixed date element of an activity cannot be verified at build-time since not all parameter values are known [19]; i.e., generally the particular date of this element is set during run-time.

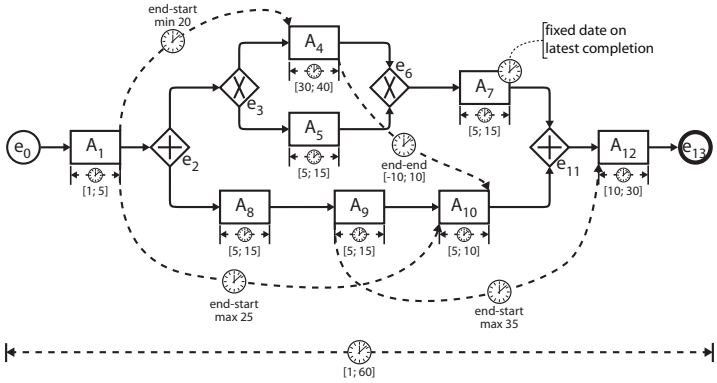

Fig. 14. Process containing different time patterns

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

5 Validation

This section discusses how we validated the described formal semantics of the time patterns against approaches from academia. The goal of this validation is to evaluate how well the described formal semantics matches the semantics inherent to these approaches. In particular, we consider the well-known proposals made by Bettini [24, 51], Combi [5, 25], Eder [7, 52], and Zhuge [53, 54]. These approaches were selected based on a systematic literature review (see [19] for details). In particular, they provide the broadest support of the time patterns, i.e., each of them considers at least 4 of the presented time patterns. Further, they provide an implementation for some of the time patterns.

For each approach we determine the impact an occurrence of a specific pattern variant has on the traces producible by a process instance that is compliant with the pattern according to the respective formalism. We then compare this impact with the defined formal semantics to determine to what degree they harmonize with each other.

5.1 Bettini et al.

Bettini et al. [24, 51] use *simple temporal problems* (STP), a special variant of *temporal constraint networks* [46], as basic formalism for verifying the consistency of the temporal perspective of a process schema. Simply spoken, a STP is a directed graph whose vertices represent variables and whose arcs correspond

to constraints between these variables. Each variable has an assigned *domain* represented by an interval on the positive integers. In turn, each arc represents a (temporal) *constraint* between the two variables it connects, which is described by an interval on the integers. A STP is denoted as *consistent* if the graph has a solution, i.e., there exist values for the variables from the associated domains such that all constraints are satisfied [46]. A process schema is now translated into a set of STPs by decomposing the process schema along its XOR-splits, i.e., each resulting STP corresponds one possible execution path (cf. Fig. 15). When transforming an execution path into a STP, each activity is decomposed into its start and end event. For each of these events a variable is added to the set of vertices. For each activity, the variables of its start and end event are connected by an arc. Further, for each control edge an arc is added between the variables corresponding to the two events connected by it. Initially, the domains of all variables as well as temporal constraints of the arcs are left undefined, i.e., the assigned interval covers the domain of all positive integers. As example for this translation consider the process depicted in the upper part of Fig. 15 and the two corresponding STPs in the lower part of the figure. Finally, loops and the dynamic setting of the parameter value of a pattern occurrence at run-time are not considered by this approach.

Regarding time patterns, the proposal made by Bettini et al. [24] considers patterns *Time Lags between two Activities* (TP1) (called "temporal distance"), and *Durations* of activities (TP2, design choice C[a]). Further, *Fixed Date Elements* for activities (TP4, design choice C[a]) (called "deadline constraint"), and *Schedule Restricted Elements* for activities (TP5, design choice C[a]) are discussed, although no details on the implementation are provided.

A **time lag between two activities** (TP1) in a process schema is translated to the STP by adding an arc between the two variables representing the events corresponding to the two activities. The temporal constraint of the arc is then set to the parameter value of the time lag (cf. Fig. 15); i.e., in case of a minimum value the lower bound of the interval is set to the respective value, whereas for a maximum value the upper bound of the interval is set. The other bound is left unrestricted (i.e., it is set to $\pm\infty$). For an interval value, both bounds of the temporal constraint are set to the respective values. According to [46], an arc $i \rightarrow j$ from variable i to j labeled with an interval d represents the constraint $\min(d) \leqslant X_j - X_i \leqslant \max(d)$; $X_i, X_j \in \mathbb{N}$ represent the values assigned to variable i and j respectively. Note that this corresponds to $X_i + \min(d) \leqslant X_j \leqslant X_i + \max(d)$, which, in turn, is equal to function compareR (cf. Definition 9).[8] Since the approach does not consider loops, only one pair of event occurrences exists. Therefore, this semantics of a time lag between activities corresponds to Pattern Semantics 1.

Similarly, an activity **duration** (TP2, design choice C[a]) in a process schema is translated to the STP by setting the parameter value of the pattern as temporal constraint for the arc between the variables representing the start and end event of the activity (cf. Fig. 15). Consequently, this semantics is equivalent to the one

[8] Using $\max(d) = \infty$ and $\min(d) = -\infty$ for minimal and maximal value, respectively.

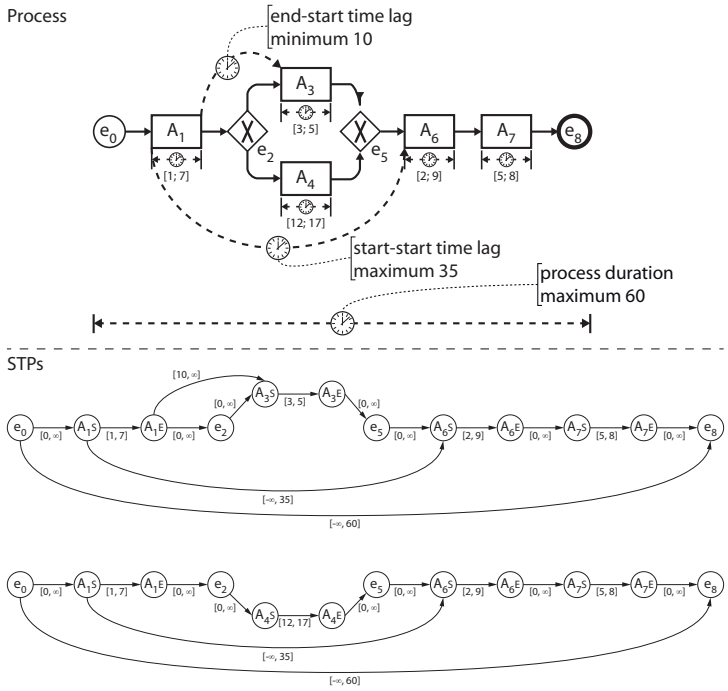

Fig. 15. A Process schema and its corresponding STPs

of Pattern Semantics 5. Other kinds of duration (i.e., activity set, process, set of process instances) are not discussed. Nevertheless, it is possible to translate the duration of a process (TP2, design choice C[c]) to a constraint between the first and last variable of the STP (similar to a time lag between activities). Again, this translation yields the same semantics as defined by Pattern Semantics 5 for this pattern variant.

An implementation of **fixed date elements** for activities (TP4, design choice C[a]) is not presented by [24], although the authors claim that they provide a way to assign a deadline to an activity. In our understanding, it is possible to implement a fixed date element by restricting the domain of the variable representing the start or end event of the activity (depending on the date restricted by the pattern occurrence, i.e., design choice F). In turn, this results in a constraint $X_i \in dom_i$, where X_i represents the value of variable i and dom_i the domain of this variable. Since dom_i is an interval, this constraint can be rewritten as $\min(dom_i) \leqslant X_i \leqslant \max(dom_i)$. Consequently, this semantics corresponds to the one of Pattern Semantics 5 (assuming that for an earliest/latest date only the minimum/maximum of the domain is set to the respective value of the pattern and the other part is left at $\pm\infty$).

Support of **schedule restricted elements** for activities (TP5, design choice C[a]) is claimed, but no implementation is described. We are not aware of any

straightforward approach to implement such a constraint based on STPs. Therefore, it is not possible to compare the semantics of the respective implementation with our formal semantics.

Time lags between events (TP3) are not considered, but may be translated to the STP the same way as TP1 (assuming that all events of the process schema are translated to variables in the STP). Again the resulting semantics corresponds to the one of Pattern Semantics 2. Finally, since dynamic changes of the parameter value of a time pattern during run-time are not considered, a **validity period** (TP7) can be implemented the same way as a fixed date element. Therefore, its semantics is equivalent to Pattern Semantics 6.

5.2 Combi et al.

The approach suggested by Combi et al. [5, 25] also uses *simple temporal problems* (STP) as formalism for verifying the consistency of the temporal perspective of a process schema. However, unary constraints (i.e., domains of variables) are represented by an arc between a special origin variable (representing time point 0) and the respective variable having the corresponding domain as temporal constraint. Again, a process schema is decomposed along its XOR-splits into a set of STPs, such that each STP represents one possible execution path (cf. Fig. 15). In particular, the transformation process is similar to the one used by Bettini et al. (cf. Sec. 5.1). Additionally, 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 iterations of the loop. Based on this, a loop can be transformed into a consecutive set of XOR-splits for deciding whether the next iteration shall be executed or remaining iterations shall be skipped (cf. Fig. 16). In turn, this can be translated into a STP the same way normal XOR-splits are handled. Furthermore, the dependencies between resulting STPs are considered, i.e., implicit constraints discovered in the STP of one execution path are propagated to the STPs of all other execution paths containing the involved variables. Finally, changing the parameter value of a pattern occurrences during run-time is not considered.

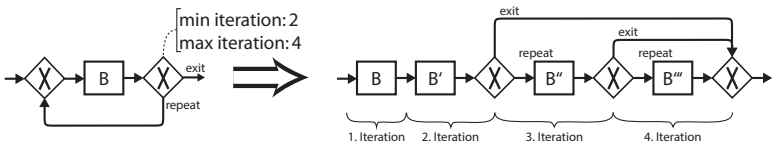

Fig. 16. Converting a Loop to a Set of Consecutive XOR-splits

In terms of the time patterns Combi et al. [5, 25] consider *Time Lags between Activities* (TP1) and *Durations* of activities (TP2, design choice C[a]). Further, *Fixed Date Elements* for the earliest start and the latest completion of activities (TP4, design choices C[a], F[a,d]) are considered. Finally, [5] mentions *Schedule*

Restricted Elements related the start of activities (TP5, design choices C[a], F[a]) and *Periodicity* (TP10). However, no implementation details are provided.

[25] discusses two different ways of defining a **time lag between two activities** (TP1): *delays* and *relative temporal constraints*. A delay corresponds to an end-start time lag between two directly succeeding activities, i.e., it represents "the duration at disposal [of the PAIS] for managing the start of the successor(s) after the end of the predecessor(s)" [25, p. 7]. In turn, a "relative [temporal] constraint limits the time distance [...] between the starting/ending instants [(i.e., events)] of two nonconsecutive [...] nodes" [25, p. 8]. Therefore, it corresponds to a time lag between two activities. Both (i.e., delay and relative temporal constraint) are represented by an interval $[MinDuration, MaxDuration]$ and translated to the STP by adding an arc with a corresponding temporal constraint between the respective events. Again, this results in constraint $MinDuration \leqslant X_j - X_i \leqslant MaxDuration$, which is equivalent to function compareR (see Definition 9). For time lags in connection with loops, [25, p. 9] defines that "tasks [(i.e., activities)] within a cycle [(i.e., loop)] may be constrained only with respect to other tasks of the same cycle and these constraints are verified each iteration of the cycle" [25, p. 9]. Therefore, inbound and outbound time lags of a loop are excluded. For all other cases, this definition yields the same semantics as defined by Pattern Semantics 1.

A **duration** of an activity (TP2, design choice C[a]) is specified in terms of an interval $[MinDuration, MaxDuration]$ and translated to the STP by adding a constraint $MinDuration \leqslant A_E - A_S \leqslant MaxDuration$, where A_S and A_E correspond to the start and end event of the activity. Consequently, this semantics is equivalent to the respective variant of pattern semantics 4. Other kinds of durations (i.e., activity set, process, set of process instances) are not discussed. Again, it is possible to translate the duration of a process to the STP yielding the same semantics as Pattern Semantics 4 (cf. Sec. 5.1).

According to [25, p. 9] an "absolute constraint represents a time interval during which an activity can be performed", i.e., it represents a **fixed date element** for the earliest start and latest completion of the activity (TP4, design choices C[a], F[a,d]). When creating a process instance, this fixed date element is converted "into bounds that are dependent from the staring instant [(i.e., event)] of the [process] instance" [25, p. 13]. This is achieved by subtracting the starting time of the process from the interval bounds and by adding a pair of arcs to the STP: one between the start event of the process and the start event of the activity, the other one between the start event of the process and the end event of the activity. Both arcs have the same converted interval as temporal constraint [25, p. 13]. Thus, an absolute constraint $[t_{start}, t_{completion}]$ is transformed to the following two constraints: $t_{start} - X_S \leqslant X_{A_S} - X_S \leqslant t_{completion} - X_S$ and $t_{start} - X_S \leqslant X_{A_E} - X_S \leqslant t_{completion} - X_S$, where X_S is the variable representing the start event of the process instance and X_{A_S}, and X_{A_E} correspond to the variables representing the start/end event of the activity. Since we know that $X_{A_S} < X_{A_E}$ holds this inequations can be rewritten to $t_{start} \leqslant X_{A_S} < X_{A_E} \leqslant t_{completion}$. The latter is equivalent with the combination of the semantics of a

fixed date element for the earliest start and a fixed date element for the latest completion of the activity as defined by Pattern Semantics 5 for the respective pattern variant. Thus, this semantics of an absolute constraint corresponds to the one of a restricted variant of pattern fixed date element (cf. Pattern Semantics 5).

"A periodic constraint represents a periodic time interval during which an activity can be started" [5, p. 7]. This is similar to a **schedule restricted element** for the earliest and latest start of an activity (TP5, design choices C[a], F[a,b]). However, no implementation details are provided by [5]. We are unaware of any straightforward approach to implement such a constraint based on STPs. Therefore, it is not possible to compare the semantics of the respective implementation to our formal semantics.

Likewise periodic activities, which are similar to the **periodicity** pattern (TP10), are mentioned by [5]. However, no details on the implementation are provided. Again, we are not aware of any straightforward way to implement this pattern using STP. Therefore, the semantics cannot be compared.

Time lags between events (TP3) are not considered, but again may be translated to the STP the same way as TP1. Again the resulting semantics corresponds to the one of Pattern Semantics 2. As discussed, a **validity period** for an activity (TP7, design choice C[a]) may be implemented based on a fixed date element. Therefore, the same statements as for fixed date elements apply. Consequently, the semantics would be the same as defined by Pattern Semantics 6.

5.3 Eder et al.

The approach suggested by Eder et al. [7, 52] uses a timed workflow graph, which constitutes an extension of the *Critical Path Method* (CPM) [55]—a well known project planning method—as basic formalism. In turn, CPM uses activity-on-node diagrams to represent and plan projects (cf. Fig. 17). Essentially, a timed workflow graph is the same as the workflow graph. Each activity of the process has a fixed duration and is augmented by two values for its earliest and latest completion time. In particular, activity durations are considered to be deterministic, i.e., activities are assumed to have the same duration for all process instances. Consequently, the time of the activity start event can be calculated based on the one of activity end event (and vice versa). Hence, the approach only considers activity end events. To handle XOR-splits and -joins, the timed workflow graph is extended in [52] by splitting up the earliest and latest completion time into the earliest/latest completion of the best/worst case (cf. Fig. 17). These four values are calculated for each activity by using a modified version of the CPM algorithm. Finally, loops and the dynamic setting of the parameter value of a pattern occurrences during run-time is not considered by this approach.

Regarding time patterns, this approach considers patterns *Time Lags between two Activities* (TP1) (called "upper and lower bound constraints"), fixed *Durations* of activities (TP2, design choice C[a]), *Fixed Date Elements* (called "deadlines") for the latest completion of activities and processes (TP4, design choices C[a,c] and

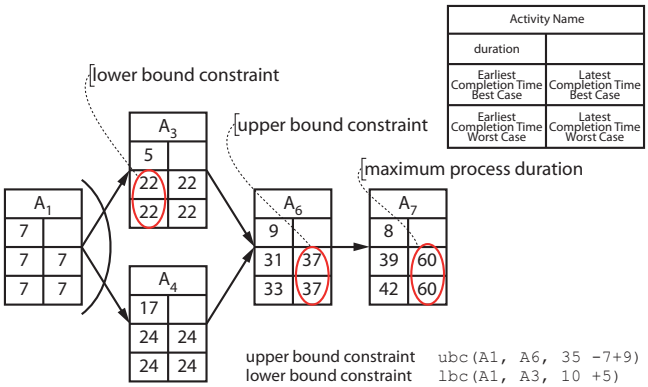

Fig. 17. Timed Workflow Graph (of the process depicted in Fig. 15)

F[d]), and *Schedule Restricted Elements* for the earliest completion of activities (TP5, design choices C[a] and F[c]) (called "fixed-date constraints").

A lower bound constraint, i.e., a minimum **time lag between two activities** (TP1, design choices D[a] and E[d]) is defined as follows: "The duration between [activity end] events A and B must be greater than or equal to δ" [7, p. 3]. Taking our notation this results in formula $\psi^t_{B_E} - \varphi^t_{A_E} \geqslant \delta$ with $\varphi^t_{A_E}$ and $\psi^t_{B_E}$ being the time of occurrence of events A and B respectively. Further, $\delta = \min(distance(\cdot, e_{A_E}, e_{B_E}, \cdot))$ corresponds to the minimum time distance between the two events, which is independent of the particular trace τ and the particular time point. In turn, an upper bound constraint, i.e., a maximum time lag between two activities (TP1, design choices D[b] and E[d]) is defined as follows: "The time distance between [activity end] events A and B must be smaller than or equal to δ"[7, p. 3]. Again, in our notation this results in formula $\psi^t_{B_E} - \varphi^t_{A_E} \leqslant \delta = \max(distance(\cdot, e_{A_E}, e_{B_E}, \cdot))$. Accordingly, this semantics of a time lag between activities corresponds to the one of the respective variant of Pattern Semantics 1. Since durations of activities are assumed to be the same for all process instances (i.e., deterministic), start-start, start-end, and end-start time lags may be also defined by adding the duration value of activity A or B or both to the respective lower or upper bound constraint. Again, under the given assumptions this results in a similar semantics as defined by Patterns Semantics 1.

Eder et al. "assume that activity durations are deterministic" [7, p. 2], i.e., the **duration** of an activity (TP2, design choices C[a]) is assumed to be the same for all process instances. This results in formula $\varphi^t_{A_S} + d_A = \psi^t_{A_E}$, where $\varphi^t_{A_S}$ and $\psi^t_{A_E}$ correspond to the time of occurrence of the start/end event of activity A, and d_A to the duration of A. In turn, this can be interpreted as an interval duration of an activity–according to pattern semantics 4–with $min(duration(\cdot, A, \cdot)) = d_A = max(duration(\cdot, A, \cdot))$. Accordingly, this semantics may be considered a restricted variant of Pattern Semantics 4.

Fixed date elements for the latest completion date of activities (TP4, design choice C[a] and F[d]) (called "deadlines") are realized by ensuring that the latest completion time of the respective activity is before or equal to the corresponding deadline [7]. This results in condition $\varphi^t_{A_E} \leqslant \mathit{deadline}$ for the time of occurrence $\varphi^t_{A_E}$ of the end event of the activity. In turn, this yields same semantics as Patterns Semantics 5 for this pattern variant. Again, since activity durations are assumed to be deterministic, a fixed date for the latest start of an activity can be specified by adding the duration of the activity to the respective fixed date. This value is then used as deadline for the latest completion of the activity. Again, this results in a restricted, but similar semantics as defined by Pattern Semantics 5.

Schedule restricted elements for the completion of activities (TP5, design choice C[a] and F[c,d]) are implemented using a so-called "fixed-date object". The latter represents "an abstraction that generalizes a typically infinite set of dates (i.e. 'every other Monday' [...])" [52, p. 7]. A fixed-date object is similar to a schedule as set out by Definition 11. Further, it has two basic methods $T.next(D)$ and $T.prev(D)$ which return the next and previous valid date of an arbitrary date D according to the fixed-date object, i.e., the start/end of the next/previous time slot. During run-time, a schedule restricted element is then implemented by setting the earliest completion time of the activity to the next date valid according to the fixed-date object. Likewise, the latest completion date is set to the previous valid date. This is similar to Patterns Semantics 8, but not completely equal as gaps between different valid dates (i.e., between the first and last valid time slot) are not fully considered. Therefore, our semantics is more complete and accurate.

As discussed, the **validity period** of an activity (TP7, design choice C[a]) can be implemented based on a fixed date element. Therefore, the same statements as for fixed date elements apply. Consequently, the semantics would be the same as defined by Pattern Semantics 6.

5.4 Zhuge et al.

Zhuge et al. [54, 53] do not use any specific formalism. Instead, temporal constraints are defined in terms of restrictions for start and completion times of activities. In turn, this is similar to the way our semantics is defined. Moreover, different time zones are considered when defining temporal constraints. However, actually, this is not necessary: on one hand, temporal constraints are generally specified with respect to a reference time zone, on the other the temporal constraints can be converted from one time zone to another (as implicitly demonstrated by [53]). Finally, neither loops nor dynamic changes of the parameter value of a pattern occurrences during run-time are considered.

Regarding the time patterns, Zhuge et al. consider patterns *Time Lags between two activities* (TP1), *Durations* of activities and processes (TP2, design choice C[a, c]), and *Fixed Date Elements* for activities (TP4, design choice C[a]) (called "fixed-point constraints").

Two different ways of specifying a **time lag between two activities** (TP1) are discussed: *flow duration* and *duration constraint* on the time between activities. A *flow duration* specifies the minimum and maximum time for passing control from activity A_i to A_j. However, it is unclear whether activities A_i and A_j must directly succeed each other. A flow duration is similar to an end-start time lag (TP1, design choice D[c] and E[c]), yet not completely equal. Note that in general, it cannot be ensured that the second activity is started immediately after receiving control. Nevertheless, its semantics is defined similarly to the interval variant of Pattern Semantics 1, i.e., a flow duration between activities A_i and A_j is defined as [53, p. 233]: $d(F_{ij}) \leqslant S(A_j) - E(A_i) \leqslant D(F_{ij})$ where $E(A_i)$ is the time of the end event of activity A_i, $S(A_j)$ is the time of the start event of activity A_j, and $d(F_{ij})$ and $D(F_{ij})$ are the minimum and maximum flow duration, respectively. In turn, a *duration constraint* restricts the maximum time between the start event of activity A_i and the end event of activity A_j. Therefore, it is similar to a start-end maximum time lag between the two activities (TP1, design choice D[b] and E[b]). It is defined by formula $E(A_j) - S(A_i) \leqslant p_j$, where $S(A_i)$ corresponds to the start of activity A_i, and $E(A_j)$ to the end of activity A_j. Further, $p_j (= \max(distance(\cdot, e_{A_iS}, e_{A_jE}, \cdot)))$ corresponds to the respective duration constraint. In turn, this is similar to a maximum start-end time lag according to Pattern Semantics 1. Consequently, the semantics of time lags between two activities considered by [53] is equal to the respective variant of our pattern semantics.

The **duration** of an activity A (TP2, design choice C[a]) is specified by a minimum value $d(A)$ and a maximum value $D(A)$. Furthermore, durations are defined by formula $d(A) \leqslant E(A) - S(A) \leqslant D(A)$, with $S(A)$ / $E(A)$ being the time of the start/end event of activity A. Consequently, this semantics corresponds to the respective variant of Pattern Semantics 4. Moreover, a minimum process duration is defined as $d \leqslant d(Wf_S)$ and a maximum one as $D(Wf_L) \leqslant D$ (TP2, design choice C[c]). Thereby, d (D) corresponds to the minimum (maximum) process duration. In turn, $d(Wf_S)$ is the minimally possible duration of the shortest process instance Wf_S. Likewise, $D(Wf_L)$ is the maximally possible duration of the longest possible process instance Wf_L. Hence, this semantics does not match the one of time pattern TP2 (cf. Pattern Semantics 4). However, the semantics defined by Zhuge et al. does not meet the intended semantics of a process duration. In particular, it factors the actual duration of a process instance out, but only considers the duration of the overall best/worst case, i.e., it only considers process instances where all activities require their minimum/maximum duration. Yet, this (almost) never happens. Consequently, the semantics defined by Zhuge et al. is too strict. Hence, we are convinced that Patterns Semantics 4 meets the intended semantics of this pattern variant.

A **fixed date element** (TP4) for the latest start of activity A_i is defined by formula $S(A_i) \leqslant p_i$, with $S(A_i)$ being the time of the start event of activity A_i and p_i being the respective date. Other variants of pattern fixed date elements (i.e., earliest start, or earliest latest completion) are defined analogously. Therefore,

this semantics is a limited variant of Pattern Semantics 5 as it does not consider changing the date value during run-time.

Again, a **validity period** for an activity (TP7, design choice C[a]) may be implemented based on a fixed date element. Therefore, same statements as for fixed date elements hold. Consequently, the semantics would be the same as defined by Pattern Semantics 6.

5.5 Summary

Our validation has proven that in most cases our pattern semantics covers the semantics of the evaluated approaches. It has further shown that the pattern semantics we defined is the most general as it covers all special cases considered by other approaches. In cases our pattern semantics differs from the one of the described approaches, we could show, that ours is more precise, meeting the intended semantics of the respective time pattern best. Finally, the introduced pattern semantics is the only one addressing the impact of dynamic changes of the parameter values of a pattern occurrence during run-time, which is important when considering run-time support.

6 Discussion

Defining a formal semantics of time patterns provides significant benefits.

First, it gives the opportunity to uncover crucial aspects that might have been neglected when only having an informal description.

Example 12 (Time Lags between Activities & Loops). *As discussed in Sec. 4.2, any semantics of the time patterns needs to be valid in connection with loops as well. Especially, this applies to time pattern TP1 (Time Lags between Activities). When formalizing this time pattern we discovered that the semantics of a time lag between two activities (TP1) of which one resides inside a loop and the other one outside that loop is unclear. Actually, there exist three different semantics (cf. Sec. 4.2). Regarding our definition of the semantics for this time pattern, we decided to use the one most obvious to process experts and not introducing any other ambiguity (cf. Sec. 4.2).*

Example 13 (Time-based Restrictions & Comparison of intervals). *When specifying the formal semantics of the time-based restrictions pattern (TP6), we had to define how many executions of an activity exist within a given time frame. Therefore, it became necessary to compare two different time intervals (i.e., the execution interval of the activity and the given time frame). However, there exist different ways to accomplish this task (e.g., subset, intersection), which is not considered by the original design choices of this time pattern. To correctly define the semantics of TP6 we added Design Choice U (cf. Fig. 12).*

If such issues are not made explicit and resolved through the definition of a precise, formal semantics, interpretations by users (e.g., system or process engineers)

might be different resulting in misunderstandings, ambiguities, and erroneous process implementations. However, we do not claim to provide the right semantics for all cases. As some time patterns carry an inherent ambiguity under certain circumstances, decisions had to be made regarding the definition of the formal semantics, which may not suit all possible scenarios (cf. Example 12). In any case, we have exposed these decisions. Additionally, due to the validation process taken, we are convinced that our semantics is the one expected by most process engineers and domain experts.

Second, the formal semantics enables us to precisely answer open questions regarding the interpretation of certain constellations of process elements. For example, if the parameter value of a time pattern may be modified during run-time, it might be unclear which of these values must be considered for a particular pattern instance (especially so in connection with concurrency and loops). By utilizing the defined pattern semantics, such questions can now be precisely answered.

Third, it becomes possible to compare and evaluate process modeling languages and PAIS according to formally defined criteria. This is additionally fostered by the fact that the provided formal semantics is independent of any particular process modeling language or paradigm. Therefore, it becomes possible to compare different systems and languages with respect to their support of the temporal perspective, e.g., to identify the one most suitable in a given application context. Furthermore, any formal semantics provides a foundation for reasoning about modeling language specifications and for deriving consequences from them. In turn, the latter is an important aid in validating and evaluating proposed implementations. Formal semantics also provides a view of modeling languages that abstract from unimportant details of syntax and presentation. In particular, this view makes it possible to compare languages, and to identify additionally required language elements.

Furthermore, having a precise and formal semantics allows implementing techniques for checking conformance of process instances in respect to a process schema and its time constructs, i.e., answering the question "Do the log of the process instance and the model conform to each other?" [23]. Again, this is fostered by the fact that the formal pattern semantics we provide is defined based on *temporal execution traces*, which are closely related to process instance logs. Hence, it is possible to implement conformance checking algorithms based on the defined formal semantics of the time patterns.

Similarly, a precise, formal semantics of time patterns is a key requirement for the formal verification of the temporal perspective of processes during build-time as well as for its verification and monitoring during run-time. In particular, a formal semantics allows us to address some open issues that need to be solved before developing comprehensive verification procedures (cf. Example 12). This is especially true for more complex time patterns as well as for the link between time patterns and loops. Up to now, these issues have been avoided by excluding both complex time patterns and loops from respective considerations. However, without these constructs no complete specification of the temporal perspective is possible.

In turn, without a formal basis no universally valid verification of the temporal perspective is possible due to ambiguities and unclear semantics. Consequently, existing approaches may benefit from the formal semantics presented in this paper.

What has not been discussed in this paper is the run-time support of the temporal perspective. When executing process instances containing temporal constraints, challenging issues emerge, e.g., "How can temporal constraints be enforced?" or "What happens in case a temporal constraint is in danger of being violated or has already been violated?". Further, during run-time, one must differentiate between temporal constraints to be met in any case (e.g., delivery deadlines) and the ones that merely represent a recommendation and hence may be ignored if desired (e.g., most duration constraints). Such issues are out of scope of this paper. Yet, we are convinced that the presented formal semantics of the time patterns serves as a good basis for investigating these issues. To the best of our knowledge, it is the first attempt to precisely and formally define the semantics of the time patterns.

7 Related Work

Time patterns have been (informally) introduced in [18, 19] and evaluated in [19] (cf. Sec. 2.2 and 3.1). Barba et al. [56] applies them in the context of scheduling support for declarative workflows. However, no formal semantics has been provided so far, even though this is crucial for 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 [10]. Based on respective patterns, the expressiveness of different process meta models and process modeling tools can be compared. Further patterns were introduced including data flow patterns [11], resource patterns [12], exception handling patterns [15], activity patterns [14], user interface transformation patterns [57, 58], and process change patterns [16, 13, 17]. Most of them come along with a formal semantics, for example, based on languages such as petri nets [10] or pi-calculus [42].

Existing approaches dealing with the temporal perspective in PAISs largely focus on specific time features like the verification of temporal constraints [5, 6, 7, 24, 25, 53] (cf. Sec. 5), escalation management [59], and scheduling support [56, 60, 61]. [62] investigates the mutual dependencies between different temporal constraints and derives inference rules for their verification and monitoring. Additionally, time support features like process monitoring [63], process mining [64], and the effect of ad-hoc changes on temporal constraints [65] have been studied. A systematic elaboration of the requirements for time support as well as respective formal semantics has been missing so far.

A topic not considered in this paper concerns time granularities and their application in the context of the time patterns. Reasoning about relative temporal constraints and time granularities has been extensively discussed in literature [66, 67, 68, 69]. Especially, it becomes relevant when verifying the consistency

and satisfiability of the temporal constraints of a process schema at build-time. It can be shown that it is an NP-hard problem to decide whether an arbitrary set of temporal constraints with granularities is consistent [67]. For defining the time pattern semantics, however, this poses no problem. Since respective pattern semantics is defined based on temporal execution traces, the absolute time points of all events are known. Thus, only these absolute time points have to be compared with the respective relative time distance. In particular, no comparison between different relative time distances is required. For this reason, time granularities were not considered in this paper.

8 Summary and Outlook

We have specified the formal semantics of the time patterns introduced in [18, 19]. Their initial introduction complemented existing workflow patterns. In particular, time patterns allow for a more meaningful evaluation of existing PAISs in case temporal constraints are 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 provides 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 has been specified based on traces in a language-independent manner. Our future work will include time patterns for aspects other than control flow (e.g., data or process changes). Furthermore, we will conduct a comprehensive study on time support features, e.g., enabling the verification of temporal constraints, escalation management, and advanced scheduling support. We will consider the resource dimension in this context as well. Finally, we will evaluate the impact process changes have on the time patterns and respective temporal specifications of business processes.

References

1. Reichert, M., Weber, B.: Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies. Springer Berlin / Heidelberg (2012)
2. Weber, B., Reichert, M., Wild, W., Rinderle-Ma, S.: Providing integrated life cycle support in process-aware information systems. International Journal of Cooperative Information Systems **18**(1) (March 2009) 115–165
3. Lenz, R., Reichert, M.: It support for healthcare processes - premises, challenges, perspectives. Data & Knowledge Engineering **61**(1) (April 2007) 39–58
4. Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT support for release management processes in the automotive industry. In Dustdar, S., Fiadeiro, J.L., Sheth, A., eds.: Proceedings of the 4th International Conference on Business Process Management (BPM'06). Volume 4102 of Lecture Notes in Computer Science., Vienna, Austria, Springer Berlin / Heidelberg (September 2006) 368–377
5. Combi, C., Gozzi, M., Juarez, J.M., Oliboni, B., Pozzi, G.: Conceptual modeling of temporal clinical workflows. In Goranko, V., Wang, X.S., eds.: Proceedings of

the 14th International Symposium on Temporal Representation and Reasoning (TIME'07), Alicante, Spain, IEEE Computer Society Press (June 2007) 70–81
6. Marjanovic, O., Orlowska, M.E.: On modeling and verification of temporal constraints in production workflows. Knowledge and Information Systems **1**(2) (1999) 157–192
7. Eder, J., Panagos, E., Rabinovich, M.: Time constraints in workflow systems. In Jarke, M., Oberweis, A., eds.: Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99). Volume 1626 of Lecture Notes in Computer Science., Heidelberg, Germany, Springer Berlin / Heidelberg (June 1999) 286–300
8. Dadam, P., Reichert, M., Kuhn, K.: Clinical workflows - the killer application for process-oriented information systems. In: Proceedings of the 4th International Conference on Business Information Systems (BIS'00). (2000) 36–59
9. Mutschler, B., Weber, B., Reichert, M.: Workflow management versus case handling: results from a controlled software experiment. In: Proceedings of the 2008 ACM symposium on Applied computing (SAC'08), New York, NY, USA, ACM Press (2008) 82–89
10. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51
11. Russell, N.C., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow data patterns: Identification, representation and tool support. In Delcambre, L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, O., eds.: Proceedings of the 24th International Conference on Conceptual Modeling (ER'05). Volume 3716 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2005) 353–368
12. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow resource patterns: Identification, representation and tool support. In Pastor, O., e Cunha, J.F., eds.: Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE'05). Volume 3520 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2005) 216–232
13. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data & Knowledge Engineering **66**(3) (2008) 438–466
14. Thom, L., Reichert, M., Iochpe, C.: Activity patterns in process-aware information systems: Basic concepts and empirical evidence. International Journal of Business Process Integration and Management **4**(2) (2009) 93–110
15. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Exception handling patterns in process-aware information systems. Technical report, BPMCenter.org (2006)
16. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In Krogstie, J., Opdahl, A.L., Guttorm, S., eds.: Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE'07). Volume 4495 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (June 2007) 574–588
17. Rinderle-Ma, S., Reichert, M., Weber, B.: On the formal semantics of change patterns in process-aware information systems. In Li, Q., Spaccapietra, S., Yu, E., Olivé, A., eds.: Proceedings of the 27th International Conference on Conceptual Modeling (ER'08). Volume 5231 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2008) 279–293
18. Lanz, A., Weber, B., Reichert, M.: Workflow time patterns for process-aware information systems. In Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper,

E., Schmidt, R., Ukor, R., eds.: Proceedings of the 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010. Volume 50 of Lecture Notes in Business Information Processing., Hammamet, Tunesia, Springer Berlin / Heidelberg (June 2010) 94–107
19. Lanz, A., Weber, B., Reichert, M.: Time patterns for process-aware information systems. Requirements Engineering (2012) (online first).
20. www.timepatterns.org: Time patterns for process-aware information systems (Last access 20.12.2012).
21. Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P., Mulyar, N.: Workflow control-flow patterns: A revised view. Technical Report BPM-06-22, BPMCenter.org (2006)
22. van Glabbeek, R., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Informatica **37** (2001) 229–327
23. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1) (2008) 64–95
24. Bettini, C., Wang, X.S., Jajodia, S.: Temporal reasoning in workflow systems. Distributed and Parallel Databases **11**(3) (2002) 269–306
25. Combi, C., Gozzi, M., Posenato, R., Pozzi, G.: Conceptual modeling of flexible temporal workflows. ACM Transactions on Autonomous and Adaptive Systems **7**(2) (July 2012) 19:1–19:29
26. Reichert, M., Dadam, P.: Towards process-oriented hospital information systems: Some insights into requirements, technical challenges and possible solutions. In: Proceedings of the 43. Jahrestagung der GMDS. (1998) 175–180
27. Schultheiß, B., Meyer, J., Mangold, R., Zemmler, T., Reichert, M.: Prozessentwurf für den ablauf einer stationären chemotherapie. Technical Report DBIS-5, Universität Ulm (1996) (in german).
28. Schultheiß, B., Meyer, J., Mangold, R., Zemmler, T., Reichert, M.: Prozessentwurf am beispiel eines ablaufs einer op. Technical report, Universität Ulm (1996) (in german).
29. Käfer, R.: Medical information processing in the hospital - current status, problems and perspectives by the example of the medical university clinic ulm. Diploma thesis, Heidelberg University (1993)
30. Kuhn, K., Reichert, M., Käfer, R., Köhler, C.: Situations- und Schwachstellenanalyse der Informationsverarbeitung in einer Universitätsklinik. In: Proceedings 39. Jahrestagung der GMDS. (1994)
31. Li, C.: Mining Process Model Variants: Challenges, Techniques, Examples. Phd thesis, University of Twente, The Netherlands (2010)
32. Li, C., Reichert, M., Wombacher, A.: The MinAdept clustering approach for discovering reference process models out of process variants. International Journal of Cooperative Information Systems **19**(3 & 4) (2010) 159–203
33. Li, C., Reichert, M., Wombacher, A.: Mining business process variants: Challenges, scenarios, algorithms. Data & Knowledge Engineering **70**(5) (May 2011) 409–434
34. Bobrik, R.: Konfigurierbare Visualisierung komplexer Prozessmodelle. PhD thesis, University of Ulm (2008)
35. German Association of the Automotive Industry (VDA): Engineering change management. part 1: Engineering change request (ECR) (2005)
36. Federal Aviation Administration: Aviation maintenance technician handbook, chapter 8. inspection fundamental. `http://www.faa.gov/library/manuals/aircraft/amt\_handbook/media/FAA-8083-30\_Ch08.pdf` (2008) (accessed 29.04.2011).

37. IACA (International Air Carrier Association): Subpart Q - flight and duty time limitations and rest requirements. `http://www.iaca.be/iaca/library/q15922\_3.pdf` (2004) (accessed 29.04.2011).
38. Stoicsics, M.: 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 (November 2011)
39. Steinle, M.: Use of workflow engines for efficient adaptation of standardized industry software to different customer requirements (in german). Masters thesis, Ulm University (2005)
40. Grambow, G., Oberhauser, R., Reichert, M.: Event-driven exception handling for software engineering processes. In Daniel, F., Barkaoui, K., Dustdar, S., eds.: Proceedings of the 5th International Workshop on event-driven Business Process Management (edBPM'11). Volume 99 of Lecture Notes in Business Information Processing., Clermont-Ferrand, France, Springer Berlin / Heidelberg (August 2012) 414–426
41. Hallerbach, A.: Management von Prozessvarianten. Phd thesis, University of Ulm (2010)
42. Puhlmann, F., Weske, M.: Using the pi-calculus for formalizing workflow patterns. In van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Proceedings of the 3rd International Conference on Business Process Management (BPM'05). Volume 3649 of Lecture Notes in Computer Science., Nancy, France, Springer Berlin / Heidelberg (September 2005) 153–168
43. Huth, M., Ryan, M.: Logic in Computer Science - modelling and reasoning about systems. Cambridge University Press (2004)
44. Cerone, A., Maggiolo-Schettini, A.: Time-based expressivity of time petri nets for system specification. Theoretical Computer Science **216**(1-2) (1999) 1 – 53
45. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126**(2) (1994) 183 – 235
46. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49** (1991) 61–95
47. Lockyer, K.G., Gordon, J.: Project Management and Project Network Techniques. Prentice Hall (2005)
48. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through sese decomposition. In Krämer, B., Lin, K.J., Narasimhan, P., eds.: Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC'07). Volume 4749 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2007) 43–55
49. Terenziani, P.: Integrating calendar dates and qualitative temporal constraints in the treatment of periodic events. IEEE Transactions on Knowledge and Data Engineering **9**(5) (September 1997) 763–783
50. Anselma, L.: Recursive representation of periodicity and temporal reasoning. In Combi, C., Ligozat, G., eds.: Proceedings of the 11th International Symposium on Temporal Representation and Reasoning (TIME'04), Tatihou, Normandie, France, IEEE Computer Society Press (July 2004) 52–59
51. Bettini, C., Wang, X.S., Jajodia, S.: Free schedules for free agents in workflow systems. In: Proceedings of the 7th International Workshop on Temporal Representation and Reasoning (TIME'00). (2000)
52. Eder, J., Panagos, E.: Managing time in workflow systems. In Fischer, L., ed.: Workflow Handbook 2001. Future Strategies Inc. (2000) 109–132

53. Zhuge, H., Cheung, T.Y., Pung, H.K.: A timed workflow process model. Journal of Systems and Software **55**(3) (2001) 231–243
54. Zhuge, H., Pung, H.K., Cheung, T.Y.: Timed workflow: Concept, model, and method. In Zhou, X., Fong, J., Jia, X., Kambayashi, Y., Zhang, Y., eds.: Proceedings of the 1st International Conference on Web Information Systems Engineering. Volume 1., IEEE (2000) 183–189
55. Philipose, S.: Operations Research - A Practical Approach. Tata McGraw-Hill (1986)
56. Barba, I., Lanz, A., Weber, B., Reichert, M., Valle, C.D.: Optimized time management for declarative workflows. In Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Wrycza, S., eds.: Proceedings of the 13th International Conference BPMDS 2012, 17th International Conference EMMSAD 2012, and 5th EuroSymposium. Volume 113 of Lecture Notes in Business Information Processing., Springer Berlin / Heidelberg (June 2012) 195–210
57. Kolb, J., Hübner, P., Reichert, M.: Automatically generating and updating user interface components in process-aware information systems. In: 20th International Conference on Cooperative Information Systems. Number 7565 in Lecture Notes in Computer Science, Springer (September 2012) 444–454
58. Kolb, J., Hübner, P., Reichert, M.: Model-driven user interface generation and adaptation in process-aware information systems. Technical Report UIB-2012-04, University of Ulm, Ulm (July 2012)
59. van der Aalst, W.M.P., Rosemann, M., Dumas, M.: Deadline-based escalation in process-aware information systems. Decision Support Systems **43**(2) (2007) 492–511
60. Combi, C., Pozzi, G.: Task scheduling for a temporal workflow management system. In Pustajovsky, J., Revesz, P., eds.: Proceedings of the 13th International Symposium on Temporal Representation and Reasoning (TIME'06), Budapest, Hungary, IEEE Computer Society Press (June 2006) 61–68
61. Eder, J., Pichler, H., Gruber, W., Ninaus, M.: Personal schedules for workflow systems. In van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M., eds.: Proceedings of the 1st International Conference on Business Process Management (BPM'03). Volume 2678 of Lecture Notes in Computer Science., Eindhoven, The Netherlands, Springer Berlin / Heidelberg (June 2003) 216–231
62. Chen, J., Yang, Y.: 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
63. Sayal, M., Casati, F., Dayal, U., Shan, M.C.: Business process cockpit. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02), VLDB Endowment (2002) 880–883
64. van der Aalst, W.M.P., Pesic, M., Song, M.: Beyond process mining: From the past to present and future. In Pernici, B., ed.: Proceedings of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE'10). Volume 6051 of Lecture Notes in Computer Science., Hammamet, Tunesia, Springer Berlin / Heidelberg (June 2010) 38–52
65. Sadiq, S.W., Marjanovic, O., Orlowska, M.E.: Managing change and time in dynamic workflow processes. International Journal of Cooperative Information Systems **9**(1-2) (2000) 93–116
66. Bettini, C., Mascetti, S., Pupillo, V.: A system prototype for solving multi-granularity temporal csp. In Faltings, B., Petcu, A., Fages, F., Rossi, F., eds.: Recent Advances in Constraints. Volume 3419 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2005) 142–156

67. Bettini, C., Wang, X.S., Jajodia, S.: A general framework for time granularity and its application to temporal reasoning. Annals of Mathematics and Artificial Intelligence **22**(1-2) (1998) 29–58
68. Bettini, C., Wang, X.S., Jajodia, S.: Solving multi-granularity temporal constraint networks. Artificial Intelligence **140**(1-2) (2002) 107–152
69. Combi, C., Franceschet, M., Peron, A.: Representing and reasoning about temporal granularities. Journal of Logic and Computation **14**(1) (2004) 51–77

Liste der bisher erschienenen Ulmer Informatik-Berichte

Einige davon sind per FTP von `ftp.informatik.uni-ulm.de` erhältlich
Die mit * markierten Berichte sind vergriffen

List of technical reports published by the University of Ulm

Some of them are available by FTP from `ftp.informatik.uni-ulm.de`
Reports marked with * are out of print

91-01 *Ker-I Ko, P. Orponen, U. Schöning, O. Watanabe*
Instance Complexity

*91-02** *K. Gladitz, H. Fassbender, H. Vogler*
Compiler-Based Implementation of Syntax-Directed Functional Programming

*91-03** *Alfons Geser*
Relative Termination

*91-04** *J. Köbler, U. Schöning, J. Toran*
Graph Isomorphism is low for PP

91-05 *Johannes Köbler, Thomas Thierauf*
Complexity Restricted Advice Functions

*91-06** *Uwe Schöning*
Recent Highlights in Structural Complexity Theory

*91-07** *F. Green, J. Köbler, J. Toran*
The Power of Middle Bit

*91-08** *V.Arvind, Y. Han, L. Hamachandra, J. Köbler, A. Lozano, M. Mundhenk, A. Ogiwara, U. Schöning, R. Silvestri, T. Thierauf*
Reductions for Sets of Low Information Content

*92-01** *Vikraman Arvind, Johannes Köbler, Martin Mundhenk*
On Bounded Truth-Table and Conjunctive Reductions to Sparse and Tally Sets

*92-02** *Thomas Noll, Heiko Vogler*
Top-down Parsing with Simulataneous Evaluation of Noncircular Attribute Grammars

92-03 *Fakultät für Informatik*
17. Workshop über Komplexitätstheorie, effiziente Algorithmen und Datenstrukturen

*92-04** *V. Arvind, J. Köbler, M. Mundhenk*
Lowness and the Complexity of Sparse and Tally Descriptions

*92-05** *Johannes Köbler*
Locating P/poly Optimally in the Extended Low Hierarchy

*92-06** *Armin Kühnemann, Heiko Vogler*
Synthesized and inherited functions -a new computational model for syntax-directed semantics

*92-07** *Heinz Fassbender, Heiko Vogler*
A Universal Unification Algorithm Based on Unification-Driven Leftmost Outermost Narrowing

*92-08** *Uwe Schöning*
On Random Reductions from Sparse Sets to Tally Sets

*92-09** *Hermann von Hasseln, Laura Martignon*
Consistency in Stochastic Network

92-10 *Michael Schmitt*
A Slightly Improved Upper Bound on the Size of Weights Sufficient to Represent Any Linearly Separable Boolean Function

92-11 *Johannes Köbler, Seinosuke Toda*
On the Power of Generalized MOD-Classes

92-12 *V. Arvind, J. Köbler, M. Mundhenk*
Reliable Reductions, High Sets and Low Sets

92-13 *Alfons Geser*
On a monotonic semantic path ordering

*92-14** *Joost Engelfriet, Heiko Vogler*
The Translation Power of Top-Down Tree-To-Graph Transducers

93-01 *Alfred Lupper, Konrad Froitzheim*
AppleTalk Link Access Protocol basierend auf dem Abstract Personal Communications Manager

93-02 *M.H. Scholl, C. Laasch, C. Rich, H.-J. Schek, M. Tresch*
The COCOON Object Model

93-03 *Thomas Thierauf, Seinosuke Toda, Osamu Watanabe*
On Sets Bounded Truth-Table Reducible to P-selective Sets

93-04 *Jin-Yi Cai, Frederic Green, Thomas Thierauf*
On the Correlation of Symmetric Functions

93-05 *K.Kuhn, M.Reichert, M. Nathe, T. Beuter, C. Heinlein, P. Dadam*
A Conceptual Approach to an Open Hospital Information System

93-06 *Klaus Gaßner*
Rechnerunterstützung für die konzeptuelle Modellierung

93-07 *Ullrich Keßler, Peter Dadam*
Towards Customizable, Flexible Storage Structures for Complex Objects

94-01 *Michael Schmitt*
On the Complexity of Consistency Problems for Neurons with Binary Weights

94-02 *Armin Kühnemann, Heiko Vogler*
A Pumping Lemma for Output Languages of Attributed Tree Transducers

94-03 *Harry Buhrman, Jim Kadin, Thomas Thierauf*
On Functions Computable with Nonadaptive Queries to NP

94-04 *Heinz Faßbender, Heiko Vogler, Andrea Wedel*
Implementation of a Deterministic Partial E-Unification Algorithm for Macro Tree Transducers

94-05 *V. Arvind, J. Köbler, R. Schuler*
On Helping and Interactive Proof Systems

94-06 *Christian Kalus, Peter Dadam*
Incorporating record subtyping into a relational data model

94-07 *Markus Tresch, Marc H. Scholl*
A Classification of Multi-Database Languages

94-08 *Friedrich von Henke, Harald Rueß*
Arbeitstreffen Typtheorie: Zusammenfassung der Beiträge

94-09 *F.W. von Henke, A. Dold, H. Rueß, D. Schwier, M. Strecker*
Construction and Deduction Methods for the Formal Development of Software

94-10 *Axel Dold*
Formalisierung schematischer Algorithmen

94-11 *Johannes Köbler, Osamu Watanabe*
New Collapse Consequences of NP Having Small Circuits

94-12 *Rainer Schuler*
On Average Polynomial Time

94-13 *Rainer Schuler, Osamu Watanabe*
Towards Average-Case Complexity Analysis of NP Optimization Problems

94-14 *Wolfram Schulte, Ton Vullinghs*
Linking Reactive Software to the X-Window System

94-15 *Alfred Lupper*
Namensverwaltung und Adressierung in Distributed Shared Memory-Systemen

94-16 *Robert Regn*
Verteilte Unix-Betriebssysteme

94-17 *Helmuth Partsch*
Again on Recognition and Parsing of Context-Free Grammars:
Two Exercises in Transformational Programming

94-18 *Helmuth Partsch*
Transformational Development of Data-Parallel Algorithms: an Example

95-01 *Oleg Verbitsky*
On the Largest Common Subgraph Problem

95-02 *Uwe Schöning*
Complexity of Presburger Arithmetic with Fixed Quantifier Dimension

95-03 *Harry Buhrman,Thomas Thierauf*
The Complexity of Generating and Checking Proofs of Membership

95-04 *Rainer Schuler, Tomoyuki Yamakami*
Structural Average Case Complexity

95-05 *Klaus Achatz, Wolfram Schulte*
Architecture Indepentent Massive Parallelization of Divide-And-Conquer Algorithms

95-06 *Christoph Karg, Rainer Schuler*
Structure in Average Case Complexity

95-07 *P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe*
ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen

95-08 *Jürgen Kehrer, Peter Schulthess*
Aufbereitung von gescannten Röntgenbildern zur filmlosen Diagnostik

95-09 *Hans-Jörg Burtschick, Wolfgang Lindner*
On Sets Turing Reducible to P-Selective Sets

95-10 *Boris Hartmann*
Berücksichtigung lokaler Randbedingung bei globaler Zieloptimierung mit neuronalen Netzen am Beispiel Truck Backer-Upper

95-11 *Thomas Beuter, Peter Dadam*:
Prinzipien der Replikationskontrolle in verteilten Systemen

95-12 *Klaus Achatz, Wolfram Schulte*
Massive Parallelization of Divide-and-Conquer Algorithms over Powerlists

95-13 *Andrea Mößle, Heiko Vogler*
Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

95-14 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
A Generic Specification for Verifying Peephole Optimizations

96-01 *Ercüment Canver, Jan-Tecker Gayen, Adam Moik*
Formale Entwicklung der Steuerungssoftware für eine elektrisch ortsbediente Weiche mit VSE

96-02 *Bernhard Nebel*
Solving Hard Qualitative Temporal Reasoning Problems: Evaluating the Efficiency of Using the ORD-Horn Class

96-03 *Ton Vullinghs, Wolfram Schulte, Thilo Schwinn*
An Introduction to TkGofer

96-04 *Thomas Beuter, Peter Dadam*
Anwendungsspezifische Anforderungen an Workflow-Mangement-Systeme am Beispiel der Domäne Concurrent-Engineering

96-05 *Gerhard Schellhorn, Wolfgang Ahrendt*
Verification of a Prolog Compiler - First Steps with KIV

96-06 *Manindra Agrawal, Thomas Thierauf*
Satisfiability Problems

96-07 *Vikraman Arvind, Jacobo Torán*
A nonadaptive NC Checker for Permutation Group Intersection

96-08 *David Cyrluk, Oliver Möller, Harald Rueß*
An Efficient Decision Procedure for a Theory of Fix-Sized Bitvectors with Composition and Extraction

96-09 *Bernd Biechele, Dietmar Ernst, Frank Houdek, Joachim Schmid, Wolfram Schulte*
Erfahrungen bei der Modellierung eingebetteter Systeme mit verschiedenen SA/RT–Ansätzen

96-10 *Falk Bartels, Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Formalizing Fixed-Point Theory in PVS

96-11 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Mechanized Semantics of Simple Imperative Programming Constructs

96-12 *Axel Dold, Friedrich W. von Henke, Holger Pfeifer, Harald Rueß*
Generic Compilation Schemes for Simple Programming Constructs

96-13 *Klaus Achatz, Helmuth Partsch*
From Descriptive Specifications to Operational ones: A Powerful Transformation Rule, its Applications and Variants

97-01 *Jochen Messner*
Pattern Matching in Trace Monoids

97-02 *Wolfgang Lindner, Rainer Schuler*
A Small Span Theorem within P

97-03 *Thomas Bauer, Peter Dadam*
A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration

97-04 *Christian Heinlein, Peter Dadam*
Interaction Expressions - A Powerful Formalism for Describing Inter-Workflow Dependencies

97-05 *Vikraman Arvind, Johannes Köbler*
On Pseudorandomness and Resource-Bounded Measure

97-06 *Gerhard Partsch*
Punkt-zu-Punkt- und Mehrpunkt-basierende LAN-Integrationsstrategien für den digitalen Mobilfunkstandard DECT

97-07 *Manfred Reichert, Peter Dadam*
$ADEPT_{flex}$ - Supporting Dynamic Changes of Workflows Without Loosing Control

97-08 *Hans Braxmeier, Dietmar Ernst, Andrea Mößle, Heiko Vogler*
The Project NoName - A functional programming language with its development environment

97-09 *Christian Heinlein*
Grundlagen von Interaktionsausdrücken

97-10 *Christian Heinlein*
Graphische Repräsentation von Interaktionsausdrücken

97-11 *Christian Heinlein*
Sprachtheoretische Semantik von Interaktionsausdrücken

97-12 *Gerhard Schellhorn, Wolfgang Reif*
Proving Properties of Finite Enumerations: A Problem Set for Automated Theorem Provers

97-13 *Dietmar Ernst, Frank Houdek, Wolfram Schulte, Thilo Schwinn*
Experimenteller Vergleich statischer und dynamischer Softwareprüfung für eingebettete Systeme

97-14 *Wolfgang Reif, Gerhard Schellhorn*
Theorem Proving in Large Theories

97-15 *Thomas Wennekers*
Asymptotik rekurrenter neuronaler Netze mit zufälligen Kopplungen

97-16 *Peter Dadam, Klaus Kuhn, Manfred Reichert*
Clinical Workflows - The Killer Application for Process-oriented Information Systems?

97-17 *Mohammad Ali Livani, Jörg Kaiser*
EDF Consensus on CAN Bus Access in Dynamic Real-Time Applications

97-18 *Johannes Köbler,Rainer Schuler*
Using Efficient Average-Case Algorithms to Collapse Worst-Case Complexity Classes

98-01 *Daniela Damm, Lutz Claes, Friedrich W. von Henke, Alexander Seitz, Adelinde Uhrmacher, Steffen Wolf*
Ein fallbasiertes System für die Interpretation von Literatur zur Knochenheilung

98-02 *Thomas Bauer, Peter Dadam*
Architekturen für skalierbare Workflow-Management-Systeme - Klassifikation und Analyse

98-03 *Marko Luther, Martin Strecker*
A guided tour through *Typelab*

98-04 *Heiko Neumann, Luiz Pessoa*
Visual Filling-in and Surface Property Reconstruction

98-05 *Ercüment Canver*
Formal Verification of a Coordinated Atomic Action Based Design

98-06 *Andreas Küchler*
On the Correspondence between Neural Folding Architectures and Tree Automata

98-07 *Heiko Neumann, Thorsten Hansen, Luiz Pessoa*
Interaction of ON and OFF Pathways for Visual Contrast Measurement

98-08 *Thomas Wennekers*
Synfire Graphs: From Spike Patterns to Automata of Spiking Neurons

98-09 *Thomas Bauer, Peter Dadam*
Variable Migration von Workflows in *ADEPT*

98-10 *Heiko Neumann, Wolfgang Sepp*
Recurrent V1 – V2 Interaction in Early Visual Boundary Processing

98-11 *Frank Houdek, Dietmar Ernst, Thilo Schwinn*
Prüfen von C–Code und Statmate/Matlab–Spezifikationen: Ein Experiment

98-12 *Gerhard Schellhorn*
Proving Properties of Directed Graphs: A Problem Set for Automated Theorem Provers

98-13 *Gerhard Schellhorn, Wolfgang Reif*
Theorems from Compiler Verification: A Problem Set for Automated Theorem Provers

98-14 *Mohammad Ali Livani*
SHARE: A Transparent Mechanism for Reliable Broadcast Delivery in CAN

98-15 *Mohammad Ali Livani, Jörg Kaiser*
Predictable Atomic Multicast in the Controller Area Network (CAN)

99-01 *Susanne Boll, Wolfgang Klas, Utz Westermann*
A Comparison of Multimedia Document Models Concerning Advanced Requirements

99-02 *Thomas Bauer, Peter Dadam*
Verteilungsmodelle für Workflow-Management-Systeme - Klassifikation und Simulation

99-03 *Uwe Schöning*
On the Complexity of Constraint Satisfaction

99-04 *Ercument Canver*
Model-Checking zur Analyse von Message Sequence Charts über Statecharts

99-05 *Johannes Köbler, Wolfgang Lindner, Rainer Schuler*
Derandomizing RP if Boolean Circuits are not Learnable

99-06 *Utz Westermann, Wolfgang Klas*
Architecture of a DataBlade Module for the Integrated Management of Multimedia Assets

99-07 *Peter Dadam, Manfred Reichert*
Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications. Paderborn, Germany, October 6, 1999, GI–Workshop Proceedings, Informatik ’99

99-08 *Vikraman Arvind, Johannes Köbler*
Graph Isomorphism is Low for ZPP^{NP} and other Lowness results

99-09 *Thomas Bauer, Peter Dadam*
Efficient Distributed Workflow Management Based on Variable Server Assignments

2000-02 *Thomas Bauer, Peter Dadam*
Variable Serverzuordnungen und komplexe Bearbeiterzuordnungen im Workflow-Management-System ADEPT

2000-03 *Gregory Baratoff, Christian Toepfer, Heiko Neumann*
Combined space-variant maps for optical flow based navigation

2000-04 *Wolfgang Gehring*
Ein Rahmenwerk zur Einführung von Leistungspunktsystemen

2000-05 *Susanne Boll, Christian Heinlein, Wolfgang Klas, Jochen Wandel*
Intelligent Prefetching and Buffering for Interactive Streaming of MPEG Videos

2000-06 *Wolfgang Reif, Gerhard Schellhorn, Andreas Thums*
Fehlersuche in Formalen Spezifikationen

2000-07 *Gerhard Schellhorn, Wolfgang Reif (eds.)*
FM-Tools 2000: The 4th Workshop on Tools for System Design and Verification

2000-08 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Effiziente Durchführung von Prozessmigrationen in verteilten Workflow-Management-Systemen

2000-09 *Thomas Bauer, Peter Dadam*
Vermeidung von Überlastsituationen durch Replikation von Workflow-Servern in ADEPT

2000-10 *Thomas Bauer, Manfred Reichert, Peter Dadam*
Adaptives und verteiltes Workflow-Management

2000-11 *Christian Heinlein*
Workflow and Process Synchronization with Interaction Expressions and Graphs

2001-01 *Hubert Hug, Rainer Schuler*
DNA-based parallel computation of simple arithmetic

2001-02 *Friedhelm Schwenker, Hans A. Kestler, Günther Palm*
3-D Visual Object Classification with Hierarchical Radial Basis Function Networks

2001-03 *Hans A. Kestler, Friedhelm Schwenker, Günther Palm*
RBF network classification of ECGs as a potential marker for sudden cardiac death

2001-04 *Christian Dietrich, Friedhelm Schwenker, Klaus Riede, Günther Palm*
Classification of Bioacoustic Time Series Utilizing Pulse Detection, Time and Frequency Features and Data Fusion

2002-01 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Effiziente Verträglichkeitsprüfung und automatische Migration von Workflow-Instanzen bei der Evolution von Workflow-Schemata

2002-02 *Walter Guttmann*
Deriving an Applicative Heapsort Algorithm

2002-03 *Axel Dold, Friedrich W. von Henke, Vincent Vialard, Wolfgang Goerigk*
A Mechanically Verified Compiling Specification for a Realistic Compiler

2003-01 *Manfred Reichert, Stefanie Rinderle, Peter Dadam*
A Formal Framework for Workflow Type and Instance Changes Under Correctness Checks

2003-02 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
Supporting Workflow Schema Evolution By Efficient Compliance Checks

2003-03 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values

2003-04 *Stefanie Rinderle, Manfred Reichert, Peter Dadam*
On Dealing With Semantically Conflicting Business Process Changes.

2003-05 *Christian Heinlein*
Dynamic Class Methods in Java

2003-06 *Christian Heinlein*
Vertical, Horizontal, and Behavioural Extensibility of Software Systems

2003-07 *Christian Heinlein*
Safely Extending Procedure Types to Allow Nested Procedures as Values
(Corrected Version)

2003-08 *Changling Liu, Jörg Kaiser*
Survey of Mobile Ad Hoc Network Routing Protocols)

2004-01 *Thom Frühwirth, Marc Meister (eds.)*
First Workshop on Constraint Handling Rules

2004-02 *Christian Heinlein*
Concept and Implementation of C+++, an Extension of C++ to Support User-Defined Operator Symbols and Control Structures

2004-03 *Susanne Biundo, Thom Frühwirth, Günther Palm(eds.)*
Poster Proceedings of the 27th Annual German Conference on Artificial Intelligence

2005-01 *Armin Wolf, Thom Frühwirth, Marc Meister (eds.)*
19th Workshop on (Constraint) Logic Programming

2005-02 *Wolfgang Lindner (Hg.), Universität Ulm , Christopher Wolf (Hg.) KU Leuven*
2. Krypto-Tag – Workshop über Kryptographie, Universität Ulm

2005-03 *Walter Guttmann, Markus Maucher*
Constrained Ordering

2006-01 *Stefan Sarstedt*
Model-Driven Development with ACTIVECHARTS, Tutorial

2006-02 *Alexander Raschke, Ramin Tavakoli Kolagari*
Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethode zur Spezifikation von eingebetteten Systemen

2006-03 *Jens Kohlmeyer, Alexander Raschke, Ramin Tavakoli Kolagari*
Eine qualitative Untersuchung zur Produktlinien-Integration über Organisationsgrenzen hinweg

2006-04 *Thorsten Liebig*
Reasoning with OWL - System Support and Insights –

2008-01 *H.A. Kestler, J. Messner, A. Müller, R. Schuler*
On the complexity of intersecting multiple circles for graphical display

2008-02 *Manfred Reichert, Peter Dadam, Martin Jurisch,l Ulrich Kreher, Kevin Göser, Markus Lauer*
Architectural Design of Flexible Process Management Technology

2008-03 *Frank Raiser*
Semi-Automatic Generation of CHR Solvers from Global Constraint Automata

2008-04 *Ramin Tavakoli Kolagari, Alexander Raschke, Matthias Schneiderhan, Ian Alexander*
Entscheidungsdokumentation bei der Entwicklung innovativer Systeme für produktlinien-basierte Entwicklungsprozesse

2008-05 *Markus Kalb, Claudia Dittrich, Peter Dadam*
Support of Relationships Among Moving Objects on Networks

2008-06 *Matthias Frank, Frank Kargl, Burkhard Stiller (Hg.)*
WMAN 2008 – KuVS Fachgespräch über Mobile Ad-hoc Netzwerke

2008-07 *M. Maucher, U. Schöning, H.A. Kestler*
An empirical assessment of local and population based search methods with different degrees of pseudorandomness

2008-08 *Henning Wunderlich*
Covers have structure

2008-09 *Karl-Heinz Niggl, Henning Wunderlich*
Implicit characterization of FPTIME and NC revisited

2008-10 *Henning Wunderlich*
On span-P^{cc} and related classes in structural communication complexity

2008-11 *M. Maucher, U. Schöning, H.A. Kestler*
On the different notions of pseudorandomness

2008-12 *Henning Wunderlich*
On Toda's Theorem in structural communication complexity

2008-13 *Manfred Reichert, Peter Dadam*
Realizing Adaptive Process-aware Information Systems with ADEPT2

2009-01 *Peter Dadam, Manfred Reichert*
The ADEPT Project: A Decade of Research and Development for Robust and Fexible Process Support
Challenges and Achievements

2009-02 *Peter Dadam, Manfred Reichert, Stefanie Rinderle-Ma, Kevin Göser, Ulrich Kreher, Martin Jurisch*
Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen

2009-03 *Alena Hallerbach, Thomas Bauer, Manfred Reichert*
Correct Configuration of Process Variants in Provop

2009-04 *Martin Bader*
On Reversal and Transposition Medians

2009-05 *Barbara Weber, Andreas Lanz, Manfred Reichert*
Time Patterns for Process-aware Information Systems: A Pattern-based Analysis

2009-06 *Stefanie Rinderle-Ma, Manfred Reichert*
Adjustment Strategies for Non-Compliant Process Instances

2009-07 *H.A. Kestler, B. Lausen, H. Binder H.-P. Klenk. F. Leisch, M. Schmid*
Statistical Computing 2009 – Abstracts der 41. Arbeitstagung

2009-08 *Ulrich Kreher, Manfred Reichert, Stefanie Rinderle-Ma, Peter Dadam*
Effiziente Repräsentation von Vorlagen- und Instanzdaten in Prozess-Management-Systemen

2009-09 *Dammertz, Holger, Alexander Keller, Hendrik P.A. Lensch*
Progressive Point-Light-Based Global Illumination

2009-10 *Dao Zhou, Christoph Müssel, Ludwig Lausser, Martin Hopfensitz, Michael Kühl, Hans A. Kestler*
Boolean networks for modeling and analysis of gene regulation

2009-11 *J. Hanika, H.P.A. Lensch, A. Keller*
Two-Level Ray Tracing with Recordering for Highly Complex Scenes

2009-12 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Durchgängige Modellierung von Geschäftsprozessen durch Einführung eines Abbildungsmodells: Ansätze, Konzepte, Notationen

2010-01 *Hariolf Betz, Frank Raiser, Thom Frühwirth*
A Complete and Terminating Execution Model for Constraint Handling Rules

2010-02 *Ulrich Kreher, Manfred Reichert*
Speichereffiziente Repräsentation instanzspezifischer Änderungen in Prozess-Management-Systemen

2010-03 *Patrick Frey*
Case Study: Engine Control Application

2010-04 *Matthias Lohrmann und Manfred Reichert*
Basic Considerations on Business Process Quality

2010-05 *HA Kestler, H Binder, B Lausen, H-P Klenk, M Schmid, F Leisch (eds):*
Statistical Computing 2010 - Abstracts der 42. Arbeitstagung

2010-06 *Vera Künzle, Barbara Weber, Manfred Reichert*
Object-aware Business Processes: Properties, Requirements, Existing Approaches

2011-01 *Stephan Buchwald, Thomas Bauer, Manfred Reichert*
Flexibilisierung Service-orientierter Architekturen

2011-02 *Johannes Hanika, Holger Dammertz, Hendrik Lensch*
Edge-Optimized À-Trous Wavelets for Local Contrast Enhancement with Robust Denoising

2011-03 *Stefanie Kaiser, Manfred Reichert*
Datenflussvarianten in Prozessmodellen: Szenarien, Herausforderungen, Ansätze

2011-04 *Hans A. Kestler, Harald Binder, Matthias Schmid, Friedrich Leisch, Johann M. Kraus (eds):*
Statistical Computing 2011 - Abstracts der 43. Arbeitstagung

2011-05 *Vera Künzle, Manfred Reichert*
PHILharmonicFlows: Research and Design Methodology

2011-06 *David Knuplesch, Manfred Reichert*
Ensuring Business Process Compliance Along the Process Life Cycle

2011-07 *Marcel Dausend*
Towards a UML Profile on Formal Semantics for Modeling Multimodal Interactive Systems

2011-08 *Dominik Gessenharter*
Model-Driven Software Development with ACTIVECHARTS - A Case Study

2012-01 *Andreas Steigmiller, Thorsten Liebig, Birte Glimm*
Extended Caching, Backjumping and Merging for Expressive Description Logics

2012-02 *Hans A. Kestler, Harald Binder, Matthias Schmid, Johann M. Kraus (eds):*
Statistical Computing 2012 - Abstracts der 44. Arbeitstagung

2012-03 *Felix Schüssel, Frank Honold, Michael Weber*
Influencing Factors on Multimodal Interaction at Selection Tasks

2012-04 *Jens Kolb, Paul Hübner, Manfred Reichert*
Model-Driven User Interface Generation and Adaption in Process-Aware Information Systems

2012-05 *Matthias Lohrmann, Manfred Reichert*
Formalizing Concepts for Efficacy-aware Business Process Modeling*

2012-06 *David Knuplesch, Rüdiger Pryss, Manfred Reichert*
A Formal Framework for Data-Aware Process Interaction Models

2013-01 *Frank Kargl*
Proceedings of the 7th Workshop on Wireless and Mobile Ad-Hoc Networks (WMAN 2013) in conjunction with Networked Systems 2013

2013-02 *Andreas Lanz, Manfred Reichert, Barbara Weber*
A Formal Semantics of Time Patterns for Process-aware Information Systems

Ulmer Informatik-Berichte
ISSN 0939-5091

Herausgeber:
Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
89069 Ulm