REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 14, número 2
Págs. 122 - 140

An Expanded and Refined Catalog of Time Patterns for Workflows

Mario Sánchez[*] [‡], Jorge Villalobos[*]

Fecha de recibido: 04/10/2013 Fecha de Aprobación: 15/11/2013

Resumen

Trabajos anteriores que definieron catálogos de patrones de control, recursos y datos para workflows tuvieron un rol fundamental en la evolución de esas dimensiones dentro de lenguajes y aplicaciones. Esos patrones han sido usados para evaluar la expresividad de los lenguajes, guiar su evolución, y para establecer una terminología básica que hoy en día es compartida por la mayoría de los desarrolladores de sistemas y lenguajes para workflows. Sin embargo, aún no se han obtenido resultados comparables en la dimensión de tiempo, a pesar de la gran importancia que esta tiene en muchos workflows y procesos de negocio. Aunque recientemente fueron propuestos algunos catálogos de patrones de tiempo, estas propuestas tienen varias limitaciones en su alcance y en la precisión de las descripciones que los hacen inadecuados para tareas como evaluar lenguajes con respecto a su capacidad para soportar esos patrones. En este artículo se presenta una aproximación para enfrentar este problema: por una parte, se presenta un catálogo extendido y refinado de patrones de tiempo para workflows; por otra parte, se presenta una formalización de dichos patrones basada en cálculo de eventos y en diagramas de estados, la cual permite hacer una evaluación de la expresividad de los lenguajes con respecto a los patrones y a la dimensión de tiempo.

Palabras clave: *Workflows, Dimensión de tiempo, Catálogo de patrones*

Abstract

Previous work on control flow, resource, and data patterns has had a fundamental role in delineating the corresponding dimensions of workflow languages and applications. These patterns have been used to evaluate languages' expressiveness, and have defined a basic terminology that is now shared by most workflow developers. Recently, some time patterns catalogs have been proposed, but they have some limitations with respect to the spectrum that they cover and the preciseness of the descriptions. This makes language evaluation difficult, and

---

* Systems and Computing Engineering Department, Universidad de los Andes, Carrera 1 Este # 19A-40, Bogotá, Colombia. Email: {mar-san1, jvillalo}@uniandes.edu.co

‡Se concede autorización para copiar gratuitamente parte o toda el material publicado en la *Revista Colombiana de Computación* siempre y cuando las copias no sean usadas para fines comerciales, y que se especifique que la copia se realiza con el consentimiento de la *Revista Colombiana de Computación*

the fact stands that most workflow languages are still very limited in their capacity to describe this dimension in spite of the important role that it plays in many business processes. In this paper we address both these problems by proposing an extended catalog of time patterns, and proposing a formalization for said patterns based on event calculus and state charts.

Keywords: *Workflows, Time Dimension, Pattern Catalog*

1. Introduction

In many business processes, the time dimension plays a critical role. For example, processes performed by governmental organizations usually have limitations on their durations established not only in the process description, but also in the law. However, in current workflow specification languages -- WfSL and workflow management systems -- WfMS, the elements to describe the time dimension are not very expressive and are often considered secondary. Time is not yet a first-class citizen in BPM standards [1]. A common vocabulary to name time structures in workflows is also lacking, and it is currently difficult to compare the expressiveness of workflow specification languages with respect to time.

In the recent past, the definition of patterns and the conceptualization of elements in some workflow dimensions fostered the maturation of WfSL and WfMS with respect to those dimensions. The most representative example of this is the list of control flow patterns proposed by the *`Workflow Patterns Initiative'* [2], and then revised in [3]. This list has become the standard to compare WfSLs, and many developments in the area have started as attempts to support them better. Similarly, the *`Workflow Patterns Initiative'* produced lists of *resource* patterns [4], *data* patterns [5], and *exception handling* patterns [6].

Recently, two catalogs of patterns that focus on the time dimension where released ([7], [8]) and we believe that they are an important step in the right direction. However, these catalogs are still immature and should not be considered definitive. For example, although both catalogs target the same concerns, they include different patterns and there are some aspects covered in ones that are not covered in the other. The terminology used is also different and there are several terms used for the same concept. Furthermore, these catalogs are very informal, thus making it difficult to evaluate whether a WfSL does really support each pattern.

This paper presents two main contributions. The first one is a revised catalog of workflow time patterns, which includes a formalization that should be useful to better evaluate whether languages support the patterns or not. The formalization has been constructed using Event Calculus [9] and state charts. The proposed catalog is also more structured than the previous ones, and in the process of revising the taxonomy we were able to discover several patterns that were overlooked by previous proposals. Moreover, we have compared the previous catalogs with the one we are proposing and we discovered that every "old" pattern has been included. The second contribution of this work is an updated ontology of time-related concepts, which provides the vocabulary that has been used to describe the time patterns.

The structure of the paper is as follows. Section 2 presents background work on time management that inspired the work presented in this paper. Section 3 presents the basic details about Event Calculus while the following section presents a conceptual model of the time dimension. Section 5 presents the structure of the expanded catalog of time patterns, the criteria to classify them, and some selected patterns. Finally, section 6 discusses the support for the proposed patterns[+] offered by current WfSL, discusses the matches between the work presented and the previous catalogs, and concludes the paper.

2. Background: workflows and time patterns

In the past, various works have addressed the time dimension in workflows, have identified the relevance of time patterns and constraints, and have proposed classification criteria. The classification of time constraints presented in [11] first separates structural constraints from explicit constraints. The former are restrictions reflecting the control structure of the workflow. They are categorized into deadlines and durations, and they are directly associated to tasks in a process. The latter are constraints representing temporal relations between events generated when tasks in a process are started or are completed. Explicit constraints are categorized into upper-bound, lower-bound, and fixed-date. Similar classifications of time constraints are also presented in [12], [13], [14], [15], and [16]. All these classifications tightly couple the time constraints and the control dimension. In particular, they depend on the starting/ending of activities, thus making it difficult to model time constraints based on other kinds of events or other workflow dimensions, like resources or data.

+For reasons of space, only a handful of patterns are presented in full detail in this section, but the specification for the rest is available in [10].

Two other catalogs of patterns were published more recently and around the same time, and they improve on those existing catalogs. The work of Lanz, Weber and Reichert [7] presents 10 patterns that were systematically identified with the analysis of real process models in a number of industries. They also propose a number of questions that serve to make each time pattern more specific. For example, in the time pattern *TP1: Time Lags Between Events* they propose the questions “What kinds of time lags are used?” and “How are time lags processed?” They also evaluated the support for their patterns in a number of tools such as calendar systems, project management tools, and BPM environments, and concluded that most patterns are not supported even though they appear in real models. However, the precise mechanism used to evaluate the support is not clear, since their patterns are not precisely defined.

On the other hand, the catalog of Niculae [8] was created by doing a quantitative literature review of time constraints patterns. Consequently, this catalog is shorter and is more focused on patterns with a smaller granularity. Furthermore, the foundation of Niculae's work is YAWL [17], and thus patterns are interpreted and presented from the point of view of WF-Nets.

3. Event Calculus

Because the initial list of *control flow* patterns was informally described, some patterns were frequently misinterpreted. To solve this situation, the patterns were formalized using Pi-Calculus [18] and Petri nets [19], resulting in unambiguous descriptions. As a result, it is now easier (although not fail-proof) to evaluate WfSLs with respect to the control flow patterns.

In a similar way, it would be desirable to have a formalization of the time patterns in order to avoid all misinterpretations, and provide a solid base for tool or language evaluation. To achieve this goal we studied some of the available mechanisms, and we selected *Event Calculus* -- EC as the language to describe with precision the intent of each time pattern. The EC, is a logical formalism for representing action and reasoning about their effects [9]. EC can be expressed as a first-order predicate calculus with circumscription, and it serves to infer what is true given “*what happens when*” and “*what actions do.*” EC is adequate for our purposes because it offers a suitable mechanism to describe things happening at specific points in time, given some conditions.

Simple EC is based on four elements: *fluents*, *actions* or *events*, *time points*, and some basic predicates. A fluent is a value that can vary over time such as a number (e.g. the number of users connected to a system) or a truth value (e.g., the maximum number of users has been reached). Actions are things that occur at some time point and may change the value of fluents. The basic predicates of EC are the following:

- *Initiates*(α, β, τ): Fluent β starts to hold after action α at time τ.
- *Terminates*(α, β, τ): Fluent β ceases to hold after action α at time τ.
- *InitiallyP*(β): Fluent β holds from time 0.
- $\tau1 < \tau2$: Time point $\tau1$ is before time point $\tau2$.
- *Happens*(α, τ): Action α occurs at time τ.
- *HoldsAt*(β, τ): Fluent β holds at time τ.
- *Clipped*($\tau1$,β, $\tau2$): Fluent β is terminated between times $\tau1$ and $\tau2$.

In EC, the evolution of a fluent is formalized with formulae that indicate which actions, and given which conditions, can alter its value. For example, the formulae

Initiates(*EnableActivity*,*activityIsEnabled*,τ)
HoldsAt(*processCompleted*,τ) [*Happens*(*ActivityEnabled*,τ)]

expresses that the fluent *activityIsEnabled* becomes true after action *EnableActivity*, provided that the fluent *processCompleted* was not holding at the time. In order to simplify some of the formulae and be able to work with time and time lapses, we complemented the basic elements of EC with variables and functions about time.

We used EC to formalize the time patterns for workflows. For each pattern, we defined a set of fluents that reflect the possible states of the constraint; then, we wrote the set of formulae that establish the initial conditions of these fluents and the conditions to alter their value. Furthermore, we specified an algorithm to convert these EC definitions into equivalent statecharts, which are more succinct and greatly improve communication.

4. Modeling the time dimension

Based on some of the works previously discussed ([7], [8]), the proposal made by Allen in [20], and on the Event Calculus, we have created a model of the time dimension in workflows (see figure 1). This model identifies and names the concepts that belong in this dimension, while establishing relations between them.

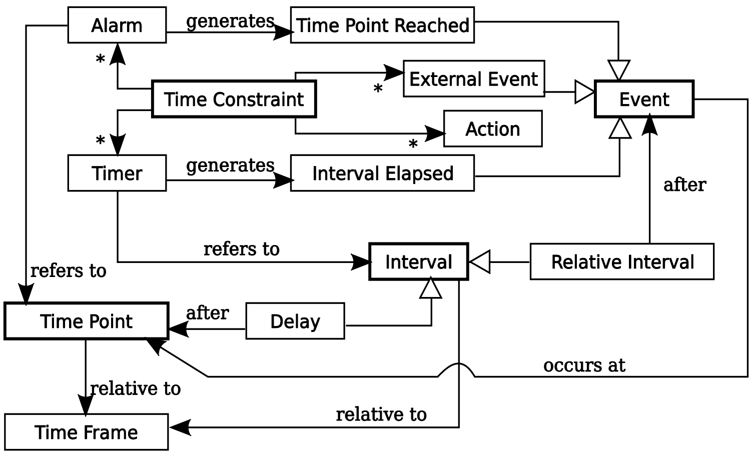

Fig. 1. A model of the time dimension in workflows.

There are four central elements in the proposed time dimension model: `Event`, `Interval`, `Time Point`, and `Time Constraint`. A `Time Point` (or fixed date) represents a precise moment of time that can be reached (e.g. 15:30:00 of November 14, 2013), and it should have a meaning within a `Time Frame` (e.g., the Gregorian Calendar, or a calendar with the workable hours of a specific company). In the catalog of Lanz et al., this is also a very important concept because employing a different reference system can modify each pattern.

An `Interval` (or time lag) represents the amount of time between two `Time Points`. `Interval`s are usually specialized using a `Time Point` and a duration (e.g., an interval of 25 minutes that starts in 15:30:00 of November 14, 2013). `Relative Interval`s do not depend on a specific `Time Point`, but depend on the occurrence of an `Event` (e.g. an interval of 25 minutes that starts after activity A1 is completed).

`Events` occur at some `Time Point`. In this dimension, there are three relevant kinds of `Events`. The first kind, `Time Point Reached`, indicates that a certain previously specified `Time Point` was reached. The second one, `Interval Elapsed`, indicates that a specified `Interval` was completed. Furthermore, there are `External Events`, which represent other events that occur in a workflow but are outside of the time domain.

`Time Constraints` are also represented in this model and each one of them can depend on the occurrence of several events of any of the three kinds. The elements responsible for generating `Time Point Reached` events are `Alarms`, and they do so at a certain `Time Points` specified in the `Time Constraint`. Similarly, `Timers` generate `Interval Elapsed` events, after a specified `Interval`. `External Events` are produced by `External Events Generators`, which are found outside of the time domain.

At last, `Time Constraints` have `Actions` associated, which define what to do if the constraints are violated or not. In this model, there are no restrictions to what an `Action` can do: its effects could have an impact on the workflow itself, or on the time constraint, or they could interact with other applications.

5. Extended Time Patterns

This section introduces the catalog of patterns that we have developed. For simplicity, we have grouped them in 11 coarse patterns, but they can be further decomposed into 42 small granularity patterns. The patterns listed in table 1 are the result of: *i*) analyzing the background work presented in section 2 (in particular the classifications introduced in [11], [12], [13], [14], [15], [16], [7], and [8]) and several current workflow specification languages; *ii*) analyzing processes from the industry and from process frameworks; *iii*) and refining and complementing a selected subset of patterns in order to have them classified in a meaningful taxonomy with as much symmetry as possible. Similarly to previous attempts, this catalog is not intended to be definitive and we expect it to be grown and improved in the same fashion as the list of control flow patterns.

The main criterion of classification within the catalog distinguishes between *time constraints* and patterns where *time is embedded in control*. The first group includes patterns that put constraints over the execution of a process, without conceptually intervening the control flow. For example, a rule specifying that a certain process should not last more than 30 days is a constraint. On the other hand, there are patterns that intervene in the control flow, for example by delaying the execution of some tasks. Moreover, this does not mean that the former patterns never interact with the control flow: when most constraints are violated, `Actions` are taken to correct or inform about the situation, or even enforce compliance, which could be considered an intromission into the control flow.

The 11 coarse patterns can be briefly described as follows:

1. **Deadline limit**: constraints about when certain events should happen with respect to a `Time Point`.
2. **Duration limit**: constraints about the duration of time lapses between events (`Interval`)
3. **Periodicity Time Limit**: constraints on the frequency of recurrent events, that is the `Interval` elapsed between subsequent occurrences of said events.
4. **Periodic Task Cardinality Limit**: constraints on the number of times recurrent events may happen within a time lapse.
5. **Simple Event Sequence**: constraints on the ordering of events that do not depend on specific intervals.
6. **Timed Event Sequence**: constraints on time lapses between the occurrences of events.
7. **Periodic Task**: control over how recurrent tasks should be executed, especially the `Interval` elapsed between subsequent executions of said tasks.
8. **Periodic Task Enforce Cardinality**: control over how recurrent tasks should be executed, especially the number of executions.
9. **Delay For**: control over when to execute some tasks, based on an `Interval.`
10. **Delay Until**: control over when to execute some tasks, based on a `Time Point`.
11. **Schedule**: control over the precise `Time Points` where tasks should be executed.

Within these patterns, we have introduced further decompositions. The first one distinguishes between *static* and *dynamic* patterns. In the former, the allowed intervals or deadlines are all known and set before the execution is started. In the latter, both intervals and deadlines depend on what happens during the execution of the processes and cannot be completely established before hand. Another kind of decomposition deals with how limits for the constraints are established and how are violated (e.g., *before* a time point, *after* a time point, or *between* two time points). The catalog of Lanz et al. considers such decompositions simple *design choices*; however, we believe that they have such an important impact that the constraints “Deadline Limit: Lower, Static” and “Deadline Limit: Excluded, Dynamic” should be considered two different patterns.

TIME CONSTRAINTS				TIME EMBEDDED IN CONTROL	
1. Deadline Limit				**7. Periodic Task**	
Lower	Upper	Inter	Excluded	n-Times	Until Event
Static		Dynamic		Static	Dynamic
2. Duration Limit				**8. Periodic Task Enforce Cardinality**	
Lower	Upper	Inter		At Least M Times Until Event	At Most M Times Until Event
Static		Dynamic		Static	Dynamic
3. Periodicity Time Limit				**9. Delay For**	
Lower	Upper	Inter		Static	Dynamic
Static		Dynamic			
4. Periodic Task Cardinality Limit				**10. Delay Until**	
Lower	Upper	Inter		Static	Dynamic
Static		Dynamic			
5. Simple Event Sequence				**11. Schedule**	
6. Timed Event Sequence				Static	Dynamic
Lower	Upper	Inter			

Table 1. Time Constraint Patterns

The rest of the section describes in detail five of the patterns. We cannot describe the full catalog in this paper, but the specification of each one of them can be found in [10]. We only include the Event Calculus specification for one of the patterns; for the rest, we provide the statechart diagrams which are equivalent and much more readable.

5.1. Deadline Limit: Inter, Static

Overview: This constraint pattern describes a situation where an event *e* must occur *between* two time points *d1* and *d2*, given that *d1 < d2*.

Motivation: This pattern captures the situations where the occurrence of an event has both lower and upper bounds. As an example, consider a process where a monthly report is always produced within the first two workable days of the following month. Therefore, the activity that produces the report has a lower bound (the first of the month) as well as an upper bound (the second day), which is precisely the situation that this pattern models.

Specification:
Parameters: This is a static pattern, which requires six parameters to be fully defined:

- *e*: the event of interest for the pattern.
- *d1*, *d2*: two time points, with *d1 < d2*.
- *A1*, *A2*, *A3*: three actions to perform depending on whether the constraint is fulfilled or not, and how is not fulfilled.

Event Calculus: We now present the EC specification of the pattern, which uses three kinds of fluents. Firstly, there are those that are true when it is not yet known whether the constraint is going to be fulfilled or not (S_0 and S_1). Then, there are those that are true while actions associated to the constraint are being executed (*Exec.A1*, *Exec.A2*, *Exec.A3*). Finally, there are those that are true when the constraint has been resolved (*EarlyEnd*, *TimeOutEnd*, *Ok End*). On the other hand, there are three kinds of events: *e*, which is the event of interest for the pattern; *e_d1* and *e_d2*, which are the events that indicate that the time points *d1* and *d2* have been reached; and *end.A1*, *end.A2*, and *end.A3* which are the events that indicate that the respective actions have completed their execution.

Table 2 presents the formulae that model this pattern. They have been grouped to improve readability and to better relate them to the ensuing statechart diagram.

$InitiallyP(S_0)$
$Happens(e\_d1, d1)$
$Happens(e\_d2, d2)$
$\neg(\exists t \mid Happens(e\_d1, t) \wedge Happens(e, t))$
$\neg(\exists t \mid Happens(e\_d2, t) \wedge Happens(e, t))$
$Initiates(e, Exec.A1, t) \leftarrow HoldsAt(S_0, t)$
$Terminates(e, S_0, t) \leftarrow HoldsAt(S_0, t)$
$Initiates(end.A1, EarlyEnd, t) \leftarrow HoldsAt(Exec.A1, t)$
$Terminates(end.A1, Exec.A1, t) \leftarrow HoldsAt(Exec.A1, t)$
$Initiates(e\_d1, S_1, t) \leftarrow HoldsAt(S_0, t)$
$Terminates(e\_d1, S_0, t) \leftarrow HoldsAt(S_0, t)$
$Initiates(e\_d2, Exec.A2, t) \leftarrow HoldsAt(S_1, t)$
$Terminates(e\_d2, S_1, t) \leftarrow HoldsAt(S_1, t)$
$Initiates(end.A2, TimeOutEnd, t) \leftarrow HoldsAt(Exec.A2, t)$
$Terminates(end.A2, Exec.A2, t) \leftarrow HoldsAt(Exec.A2, t)$
$Initiates(e, Exec.A3, t) \leftarrow HoldsAt(S_1, t)$
$Terminates(e, S_1, t) \leftarrow HoldsAt(S_1, t)$
$Initiates(end.A3, OkEnd, t) \leftarrow HoldsAt(Exec.A3, t)$
$Terminates(end.A3, Exec.A3, t) \leftarrow HoldsAt(Exec.A3, t)$

Table 2. Formulae to formalize Deadline Limit pattern.

Statechart diagram: Figure 2 shows the statechart for the Deadline Limit pattern. The expression *T(d1)* represents the event of reaching the time point *d1*. The actions *A1*, *A2*, and *A3* represent the execution of the corresponding actions associated to the time constraint.

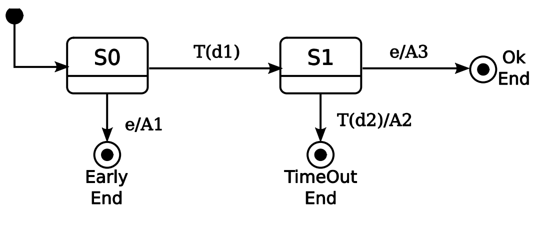

Fig. 2. Statechart for the Deadline Limit pattern.

5.2. Duration Limit: Inter, Static

Overview: This pattern describes a situation where the interval between two events *e1* and *e2* must be *longer* than *i1*, but *shorter* than *i2*. It is assumed that *e1* always occurs before *e2* and that *i1* < *i2*.

Motivation: All the *duration* patterns make reference to two events. For example, when the duration of an activity's execution has to be limited, the first event is used to indicate the start of the execution, and the second event is used to indicate its completion. In this case, the duration (or the lapse between the two events), has both lower and upper bounds. In a concrete process, this kind of time constraint can serve to ensure the quality of a tasks: if not enough time is put into it, it can be as bad as if too much time is expended in it.

Specification:
Parameters: This is a static pattern, which requires seven parameters to be fully defined:

- *e1, e2*: the two events of interest for the pattern.
- *i1, i2*: two intervals, with *i1* < *i2*.
- *A1, A2, A3*: three actions to perform depending on whether the constraint is fulfilled or not, and how is not fulfilled.

Statechart diagram:

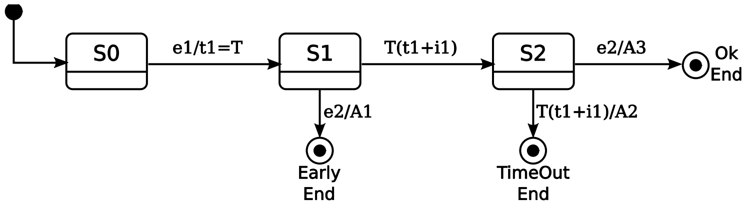

Fig. 3. Statechart for the Duration Limit pattern.

5.3. Periodic Tasks: Until Event, Static

Overview: This pattern describes a situation where some tasks have to be executed periodically, leaving intervals of length *i* between each execution. The recurrent execution of the tasks begins after an event *e1* occurs, and continues until the occurrence of the event *e2*.

Motivation: Although control flow languages typically have the capabilities to describe recurrent task sets, it is not usually possible to characterize the recurrence from the perspective of time. If such thing is desired, extraneous elements similar to *sleeps* have to be introduced in order to pause the execution until the task set has to be executed again. This pattern attempts to correct this situation by making explicit the rules that control the recurrence of the task set. As an example of the application of this pattern, one can consider a long process where someone has to receive periodical updates on its status. In this situation, the recurrent task set sends the status report, the first event signals the beginning of the process execution, the second event signals the ending of the process execution, and the interval depends on the frequency required for the updates.

Specification:
Parameters: This is a static pattern, which requires the following parameters to be fully defined:

- *e1, e2*: the two events of interest for the pattern.
- *TS*: the task set that is to be executed recurrently.
- i: an interval to leave between executions of the task set.

Statechart diagram: This statechart has an additional element with respect to the previous one: variables. In particular, the meaning of the expression *t=T+i* is that variable *t* is initialized with the current time (*T*) increased by the interval *i*. Similarly, the expression *T(t)* represents the event of reaching the time point referenced by the variable *t*.

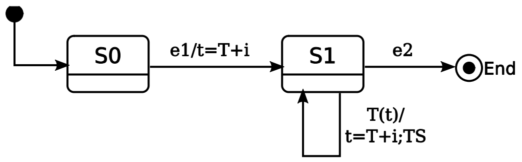

Fig. 4. Statechart for the Periodic Tasks - Stop Event pattern.

5.4. Delay Until: Static

Overview: This blocking pattern describes a situation where the execution of a task *k* should not begin before a certain time point *d*. If the task starts before that, its execution is blocked and is only allowed to continue after the point is reached.

Motivation: The control flow dimension of a process determines when a task becomes enabled to be executed, depending on the completion of other tasks. However, the conditions to enable a task sometimes include factors related to time, which have to be enforced just as control flow restrictions are enforced. This pattern serves to describe this kind of situations.

With respect to the previously described patterns, this one has major differences in being blocking. This means that the execution of this pattern has concrete consequences on the execution of the control flow. As it will be shown bellow, the specification of this pattern includes the operations *Block k* and *Continue k*, which respectively block the execution of task *k*, and enable it to continue.

Specification:
Parameters: This is a static pattern, which requires the following parameters to be fully defined:

k: the task of interest for the pattern.
d: a time point that should pass before the task is allowed to be executed.
A1, A2: actions to perform depending on whether *k* is enabled before or after the time point *d*.

Statechart diagram:

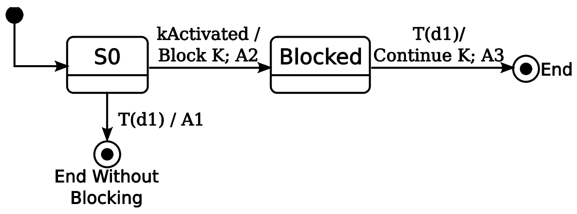

Fig. 5. Statechart for the Delay Until pattern.

5.5. Delay For: Dynamic

Overview: This blocking pattern describes the situation where the execution of task *k* has to be delayed for an amount of time *i*, that depends on the lapse between the occurrence of two events *e1* and *e2*. After the amount of time has passed, task *k* can continue its execution. This pattern assumes that the execution of *k* begins after both *e1* and *e2* have occurred, and that *e1* occurs before *e2*.

The execution of this pattern has concrete consequences on the execution of the control flow, and for this its specification includes the operations *Block k* and *Continue k*, which respectively block the execution of task *k*, and enable it to continue.

Motivation: There are two motivations behind this pattern. The first one is the idea of blocking the execution of a task for certain lapse of time, instead of blocking it until a certain time point as in Delay Until patterns. The second idea is that of making the interval unknown before execution, that is making the pattern *dynamic*. As an example of the application of this pattern, consider the case where a certain document has to be made available for a certain time before a process can continue. This is something that occurs very frequently in government-related processes, where the law itself establishes that to guarantee some fairness the amount of time used by some party sets the amount of time that another party can use.

Specification:
Parameters: This is a dynamic pattern, and thus not all of its parameters have to be known prior to execution. However, the parameters that must be known are the following:

k: the task of interest for the pattern.
e1, e2: the events of interest for the pattern, which define the interval to delay task *k*.
A1, A2: actions to perform.

Statechart diagram:

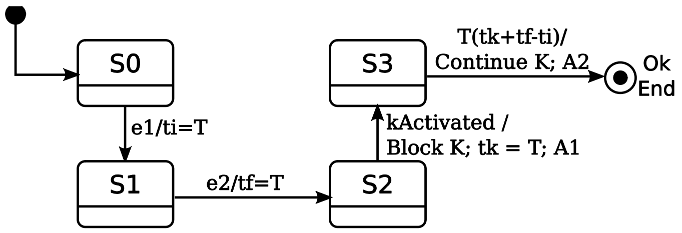

Fig. 6. Sta techart for the Dynamic Delay For pattern.

6. Time in workflow languages

Up to a certain level, the time dimension has been incorporated in both process specification languages and workflow management systems. However, there is still a lot of room to improve in this area and provide more comprehensive support [21], [15]. In particular, languages and tools currently support only a subset of the time patterns proposed in this paper, and in the other catalogs that we have mentioned already.

We performed an evaluation of BPMN with respect to our catalog in order to establish how many of the patterns can be described using this standard language for process description. The final conclusion that we reached is that only 6 patterns are *fully supported*, for 27 patterns there is *partial support*, and for 9 patterns there is *no support*. In the first group we put those patterns that can be trivially introduced in a process and have a minimal impact on the control flow. This means that it is not necessary to add complex control structures in order to support the pattern. For all the patterns in this group it is also possible to trace the execution of the process and match it to the behavior outlined in the state machine. The second group of patterns, those that are partially supported, are those where more profound changes are required on the control flow. These changes tend to increase its complexity and obfuscate the intention of the process designers, and thus are seen as undesirable. Also, some of these patterns also require relatively simple extensions to BPMN. Finally, there is a group of patterns that is not supported at all and requires the introduction of complex ad hoc extensions to the language in order to be fully supported.

There are several factors that explain why BPMN performs so poorly with respect to time patterns. The first and foremost is that the language mixes control and time, and makes the latter secondary to the first (more on this later). The second factor is that BPMN does not have an inherent way to manage variables. Therefore, *dynamic* patterns are difficult to represent, as well as those that require counting the number of repetitions. Finally, a third factor is that all expressions in BPMN related to time are fuzzy, as they have to be expressed using natural language. Therefore, expressions such as “After two hours” or “For twice the time of the original activity'” are perfectly valid although they decidedly lack precision.

The results that we obtained in the evaluation of BPMN against our catalog of time patterns are consistent with the results presented by Lanz et al. [7]. They evaluated several tools and languages, including project management and calendar systems, and both commercial and academic process languages, and found the following results: out of 10 patterns, BPMN supports only 6 and just partially. This includes the fact that, according to them, some of the patterns require a “formalism used for specifying time parameters” which is not part of BPMN, and thus “no definite conclusion can be made”.

Considering that the catalog of Lanz et al. is a subset of our own catalog, we did not replicate all the evaluations that they performed. Our hypothesis is that the results would be even worse than theirs, especially because of the additional patterns dealing with dynamicity. There has not been an evaluation of the patterns proposed by Niculae, but only one of her patterns is not included in our catalog. Therefore, we expect that any evaluation would arrive to results similar to those that have already been discussed.

The lack of capacity found in tools and languages to express time is especially caused by treating time as a secondary concern, and mixing it with the control flow. For example, BPEL [22] has a handful of elements that can be related to time, which are normal control elements, but are qualified for time using specific attributes and the element `<onAlarm>`. BPMN [23] provides three types of events to manage time in a process: a start event controlled by a timer mechanism; an intermediate event associated to the process flow and controlled by a timer; and an intermediate event associated to exception paths and controlled by a timer. In YAWL [17], the only aspect that allows the expression of time is the inclusion of an attribute in tasks to indicate a timeout. SAPWorkflow offers a similar support: tasks can have upper and lower deadlines, which the runtime system can monitor. These deadlines are defined relative to a reference point that can be the start time of a workflow or a specific date. The approach of jBPM is very different because it is based on the declaration of timers that execute actions (pieces of Java code), or enforce a process transition if a duration or date has been reached. However, this solution is very technical in its nature, and does not make it easy to do things such as reasoning about the time constraints.

There are some efforts such as [24], [1] that are trying to extend the capacity of workflow languages to express the time dimension. Also, there are tools that offer some advanced means to express and check time constraints. Among them, there is the case of ADEPT [25], which offers sophisticated modeling concepts and advanced features for temporal constraint management. It uses Temporal Constraint Networks for representing time constraints and checking their consistency. In this way, duration time constraints and deadlines can be modeled for activities and their consistency can be checked at the specification level.

7. Conclusion

Up to this point, the definition of time constraints in workflows has been secondary to the definition of control for most workflow specification languages. In some cases it is simply not possible to express, with the necessary level of detail, time requirements of the workflows; in other cases, the definition of time *follows* control and thus makes it very difficult to express time constraints in a way that does not depend too much on the control elements. Because of this, when a specific time pattern has to be expressed, the control structure has to be modified in such a complex way that its original purpose is lost.

The catalog of time patterns presented in this paper, and other similar catalogs of patterns, are intended to be an important step in the direction of offering powerful support for time in mainstream workflow languages. We expect to see, in the very near future, the evolution of languages towards supporting time in a better way, just as has happened after the introduction of catalogs of patterns for other dimensions (e.g., control, data).

The catalog presented in this paper has been developed by analyzing process catalogs as well as existing catalogs of time constraints in workflows. What we have presented here subsumes all previous work in the area and offers a further advantage: by providing a formalization of the patterns it is possible to make a more accurate evaluation about the support offered by languages or tools for each pattern. Furthermore, the mechanisms used to formalize the patterns currently in the catalog can be used as well to specify patterns that could be later included.

References

[1] D. Gagne and A. Trudel, "Time-bpmn," in *Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 361–367.

[2] W. M. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow Patterns," BPMcenter.org, Tech. Rep. BPM-03-06, 2003.

[3] N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. Mulyar, "Workflow Control-Flow Patterns: A Revised View," BPMcenter.org, Tech. Rep. BPM-06-22, 2006.

[4] Russell, Nick, van der Aalst, Wil M. P., ter Hofstede, Arthur H. M., and Edmond, David, "Workflow Resource Patterns: Identification, Representation and Tool Support," in *Lecture Notes in Computer Science*, ser. LNCS, O. Pastor and J. Falcao, Eds., vol. 3520. Springer Berlin, 2005, pp. 216–232–232.

[5] N. Russell, A. ter Hofstede, D. Edmond, and W. M. P. van der Aalst, "Workflow Data Patterns: Identification, Representation and Tool Support," in *Conceptual Modeling – ER 2005*, ser. Lecture Notes in Computer Science, L. Delcambre, C. Kop, H. Mayr, J. Mylopoulos, and O. Pastor, Eds., vol. 3716. Springer Berlin / Heidelberg, 2005, pp. 353–368.

[6] N. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede, “Exception Handling Patterns in Process-Aware Information Systems,” BPMcenter.org, Tech. Rep. BPM-06-04, 2006.

[7] A. Lanz, B. Weber, and M. Reichert, “Workflow time patterns for process-aware information systems,” in *11th International Workshop BPMDS and 15th International Conference EMMSAD at CAiSE 2010*, ser. LNBIP, vol. 50. Springer, 2010, pp. 94–107.

[8] C. Niculae, “Time Patterns in Workflow Management Systems,” Eindhoven University of Technology, Tech. Rep., 2010.

[9] M. Shanahan, *Artificial intelligence today: recent trends and developments*. Berlin, Heidelberg: Springer-Verlag, 1999, ch. The Event Calculus explained, pp. 409–430.

[10] Cumbia, “Time Patterns,” http://cumbia.uniandes.edu.co/ wikicumbia/ doku.php?id=trp:referencestimepatterns, 2013.

[11] [11] J. Eder, E. Panagos, and M. Rabinovich, “Time constraints in Workflow Systems,” in *Advances Information Systems Engineering: 11th International Conference, CAiSE99*, ser. Lecture Notes in Computer Science, vol. 1626. Springer-Verlag, 1999, pp. 286–300.

[12] O. Marjanovic and M. E. Orlowska, “On Modeling and Verification of Temporal Constraints in Production Workflows,” *Knowledge and Information Systems*, vol. 1, no. 2, pp. 157–192, 1999.

[13] C. Combi and G. Pozzi, “Temporal Conceptual Modelling of Workflows,” in *Proceedings of the 22nd Int. Conference on Conceptual Modeling ER2003*, ser. Lecture Notes in Computer Science, I.-Y. Song, S. Liddle, T.-W. Ling, and P. Scheuermann, Eds., vol. 2813. Springer, 2003, pp. 59–76.

[14] W. Sadiq, O. Marjanovic, and M. E. Orlowska, “Managing Change and Time in Dynamic Workflow Processes,” *International Journal of Cooperative Information Systems*, vol. 9, no. 1-2, pp. 93–116, 2000.

[15] H. Zhuge, T.-y. Cheung, and H.-K. Pung, “A timed workflow process model,” *Journal of Systems and Software*, vol. 55, no. 3, pp. 231–243, 2001.

[16] C. Bettini, X. S. Wang, and S. Jajodia, “Temporal Reasoning in Workflow Systems,” *Distributed and Parallel Databases*, vol. 11 no. 3, pp. 269–306, 2002.

[17] W. M. van der Aalst and A. H. M. ter Hofstede, "YAWL: Yet Another Workflow Language (Revised Version)," Queensland University of Technology, Brisbane, Tech. Rep. FIT-TR-2003-04, 2006.

[18] F. Puhlmann and M. Weske, "Using the pi-calculus for formalizing workflow patterns," in *Proceedings of 3rd International Conference on Business Process Management, BPM'05,* ser. Lecture Notes in Computer Science, van der Aalst, Wil M. P., B. Benatallah, F. Casati, and F. Curbera, Eds., vol. 3649. Springer Berlin / Heidelberg, 2005, pp. 153–168–168.

[19] L. Zhang, S. Yao, and J. Li, "Formalizing workflow patterns with Extended Petri-Net," in *Proceedings of the Sixth International Conference on Natural Computation (ICNC'2010).* IEEE, 2010, pp. 3164–3168.

[20] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM,* vol. 26, pp. 832–843, 1983.

[21] J. Eder, H. Pichler, and S. Vielgut, "Avoidance of deadline-violations for inter-organizational business processes," in *7th International Baltic Conference on Databases and Information Systems,* 2006, pp. 33–40.

[22] OASIS Technical Committee, "Web Services Business Process Execution Language, Version 2.0," http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, 2005.

[23] Object Management Group, "Business Process Model and Notation (BPMN), Beta 1 for Version 2.0," http://www.omg.org/spec/BPMN/2.0, August 2009. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

[24] D. Gagne ̀ and A. Trudel, *2009 BPM & Workflow Handbook Methods, Concepts, Case Studies and Standards in Business Process Management and Workflow.* Future Strategies Inc., 2009, ch. Extending XPDL with the Temporal Perspective.

[25] M. Reichert, S. Rinderle, and P. Dadam, "ADEPT workflow management system: flexible support for enterprise-wide business processes," in *Proceedings of the 2003 International Conference on Business Process Management - BPM03,* ser. Lecture Notes in Computer Science, vol. 2678. Springer-Verlag, 2003, pp. 370–379.