A Modular Approach to the Specification and Management of Time Duration Constraints in BPMN

Carlo Combi, Barbara Oliboni, Francesca Zerbato[*]

Department of Computer Science, University of Verona, Italy

Abstract

The modeling and management of business processes deals with temporal aspects both in the inherent representation of activities coordination and in the specification of activity properties and constraints. In this paper, we address the modeling and specification of constraints related to the duration of process activities. In detail, we consider the Business Process Model and Notation (BPMN) standard and propose an approach to define re-usable duration-aware process models that make use of existing BPMN elements for representing different nuances of activity duration at design time. Moreover, we show how advanced event-handling techniques may be exploited for detecting the violation of duration constraints during the process run-time. The set of process models specified in this paper suitably captures duration constraints at different levels of abstraction, by allowing designers to specify the duration of atomic tasks and of selected process regions in a way that is conceptually and semantically BPMN-compliant. Without loss of generality, we refer to real-world clinical working environments to exemplify our approach, as their intrinsic complexity makes them a particularly challenging and rewarding application environment.

Keywords: Business process modeling, duration constraints, BPMN, modular process design, duration patterns

1. Introduction

Business process management (BPM) focuses on the modeling and management of business processes by using suitable techniques that allow organizations to be more efficient and flexible in achieving their goals. In this context, a deep understanding of organizational processes supported by an intuitive design approach can improve the quality of the business from different viewpoints, such as costs reduction, resource planning, and increase in competitiveness.

[*]Corresponding author
Email address: `francesca.zerbato@univr.it` (Francesca Zerbato)

A very important aspect to consider when dealing with business processes is time [1, 2, 3, 4, 5]. Temporal coordination is naturally represented and managed during process design. Indeed, a business process is as a collection of activities related across *time* and space, that realizes a specific service or business goal [6]. Temporal constraints play a crucial role in process execution, as most real-world processes run under time constraints [4]. Activity performance takes time, the scheduling of resources and workforce requires temporal coordination, and process compliance with deadlines is fundamental in most application environments [7, 8, 9].

In this paper, we discuss issues related to the modeling and management of temporal duration of activities and specific process regions.

Activities are used to represent any amount of work carried out within a process, and thus, they implicitly span over a certain, finite amount of time. Explicitly representing activity duration in process models is useful whenever temporal information is crucial for understanding and re-engineering the designed procedures. Furthermore, in many real-world applications, constraining activity duration becomes essential in order to guarantee the completion of the overall process within desired time limits, the latter driven by resource availability and scheduling requirements [10]. Similarly, under a process compliance perspective, activities need to adhere to regulations and policies that set deadlines or limit duration based on best-practices. Actually, it is quite common that both the execution and results of an activity are affected by its temporal duration: some tasks and procedures become irrelevant with respect to process goals if they are not performed within predefined time limits [11].

As an example, let us consider dialysis, whose efficacy is determined by the overall duration of the treatment. Dialysis is used to purify blood from waste and extra water and a single treatment session usually lasts 3-5 hours. Despite short and rapid dialysis is less painful for the patient, rapid fluid-removal can result in depletion of the circulating volume and in hypotension. The latter, together with incomplete removal of salt and water, are associated with increased risk of death [12]. Therefore, treatments briefer than 3 hours are considered ineffective and dangerous for the patient. Such example shows that, in order to properly model activity "dialysis", the representation of defined duration constraints is an important issue to consider. Indeed, in some cases executing an activity without observing its duration constraints undermines the validity of its outcomes.

Our contribution lies on the specification and management of duration constraints using the well-known Business Process Model and Notation (BPMN) standard [13]. In general, due to the lack of *direct support* for time aspects, modeling and managing temporal constraints in BPMN is quite demanding for process designers [8]. To overcome this limitation and foster the verification of time-aware processes, a considerable number of research proposals have focused on extending the BPMN standard [5, 14, 15, 16, 17].

Compared to these approaches, our proposal starts from existing BPMN elements and allows the representation of the considered duration constraints by means of their combination. Despite requiring us some initial modeling effort, the obtained processes provide a clear conceptualization of duration-aware process activities and regions, entirely based on the standard BPMN semantics [13, 18]. This means that we design the proposed duration patterns directly with BPMN, by taking advantage of the standard semantics of BPMN itself. The designed duration patterns can also be represented by using a new kind of task, but the extension of the BPMN standard is not the focus of this work.

The process models designed in this paper are well-structured and intended to be used as base building blocks for modeling duration-aware processes. That is, we do not expect process designers to come up themselves with the proposed models, but rather provide them with ready-to-use solutions to model duration constraints in a standard-compliant way. We propose a set of modular process models that can be used as building blocks, and also define a new task type representing them. The designer can use both for specifying and managing duration constraints of tasks belonging to a BPMN process model.

Our proposal focuses on fulfilling the following research requirements.

- *Temporal Management.* In business process management, temporal aspects are a very important issue to consider. Time plays a major role in activity coordination and temporal constraints satisfaction affects process results. Thus, representing and managing temporal constraints is needed during process design and execution.
- *Standard-based modeling.* In business process representation, the use of a standard graphical notation facilitates the understanding of business procedures, internal collaborations, and coordination between activities.
- *Designer Support.* In business process modeling, designers and analysts need to be supported in order to be able to easily model business activities and the related temporal aspects.
- *Modularity/Re-usability.* In business process modeling, modularity and reusability reduce design complexity and improve both readability and management of complex process models. Having ready-to-use (sub)process models, and a new task type representing them facilitates the modeling of complex processes.

This paper deals with the specification of different kinds of duration constraints in BPMN and tackles the detection and management of constraint violations during process run-time [1, 4]. The paper comprehensively extends previous pieces of work [11, 19]. In [11], we proposed a basic process model for specifying the duration of a given activity, which is extended for constraining the duration of selected process regions and for managing duration violations. Following a similar approach, in [19] we introduced a set of BPMN models for specifying shifted durations constraints and detecting their violations.

The process model introduced in [11] is the starting point for the novel processes that are presented in this paper. As a first step forward, in this paper, we discuss the possibility of having boundary events influencing the execution of a constrained activity and propose suitable models for specifying such constraints and for discerning different causes of activity interruption. Moreover, in this paper, we introduce new simplified patterns for constraining either the minimum or the maximum duration of simple activities.

Moving from simple activities to more complex process parts, we also deal with the specification of duration constraints of Single-Entry-Single-Exit (SESE) regions, and of arbitrarily selected Non-Single-Entry-Single-Exit (non-SESE) regions [20]. Then, we address two variants of simple duration of activities: (i) one deals with the possibility of dynamically specifying activity duration after its initiation (*deferred duration*), while (ii) the other one regards *shifted duration*, i.e., a duration that is measured starting only from a relevant moment, after activity initiation [19]. To complete the picture, we introduce a new task type for representing the proposed patterns, and suitable techniques that can be embedded in the proposed process models in order to detect and handle duration violations.

In this paper, the proposed process models are described in a more formal way and an exhaustive coverage of related literature is also included. As a final step, we review our approach and discuss possible evaluation strategies to validate the execution behavior of the proposed models.

Among various domains suitable for the application of BPM techniques, clinical working environments cover a role of primary importance with respect to the temporal perspective [4, 21]. Indeed, time is extremely relevant in healthcare processes, as patients lives are involved, resources are limited, and clinical procedural aspects must be integrated with organizational and administrative practice [22]. For this reason, we often refer to real clinical examples throughout the paper, without compromising the generality of the proposed approach.

The structure of the paper is as follows. Section 2 provides the reader with basic background concepts related to BPMN and time, while Section 3 summarizes the constraints addressed in this paper. Section 4, Section 5, Section 6, and Section 7 describe the structure and behavior of the introduced process models. Section 8 presents possible strategies for detecting and handling duration violations. Section 9 provides an overview of relevant related work. Section 10 discusses the validation of our approach and outlines future work.

2. Background: BPMN, Structured Design, and Temporal Aspects

In this section, we recall the main elements of the Business Process Model and Notation [13], then we unravel the temporal character of selected BPMN elements, and discuss the principle of well-structured process design.

2.1. Introducing the Business Process Model and Notation

Among existing process modeling languages, we chose to focus on the specification of temporal constraints through BPMN, as it is the leading standard for business process modeling.

In BPMN, a process is defined as sequence of *activities* or *events*, connected by a *sequence flow* (also called *control flow*), that denotes their ordering relations. Routing is realized by *gateways*, which allow to split the sequence flow into multiple paths and merge them.

A BPMN process is visualized by means of a graphical diagram, and its behavior can be represented through tokens that traverse the process flow. In BPMN, a token is a theoretical concept that is used as an aid for defining the process behavior when it is performed. The behavior of a process element can be defined by describing how it interacts with a token as the token 'traverses' the structure of the process. Depending on the semantics of traversed flow element, the number of tokens in a process can vary, as they are continuously generated and consumed [13]. As an example, a *start event* generates a token, while an *end event* consumes one.

The basic elements (or *flow nodes*) of a BPMN process are described below.

Activities identify work that is performed within the process. In BPMN there are two kinds of activities: *tasks* that are atomic units of work that cannot be broken down to a finer level of abstraction, and *subprocesses* that are compound activities whose internal details are modeled using other elements. Graphically, tasks are depicted as rectangles with rounded corners having a label that specifies their name. Subprocesses can either be represented as collapsed, i.e., as tasks decorated by a "+" sign, or can be expanded to show internal details.

Events represent facts that occur instantaneously during process execution and that affect the sequencing or timing of process activities. They are visualized as circles, which may contain a marker to diversify the kind of event trigger. Depending on their behavior and on the type of trigger, events can either throw or catch a result. The throwing and catching of an event are referred to as *event handling* [13]. Usually the (sub)processes that focus on handling events are called "event handlers". By convention, start events initiate a process instance, end events conclude it, while intermediate events indicate where something happens somewhere between the start and end of a process. When attached to an activity boundary, *interrupting* intermediate events interrupt the task they are attached to, whereas *non-interrupting* ones initiate an new process path (*exception flow*), which runs in parallel to activity execution.

Gateways are elements in the process used to control the divergence and convergence of the sequence flow, either according to data-based conditions or event occurrence. Graphically, they are shown as diamonds with an internal marker that differentiates their routing behavior. Symbol **+** denotes parallel gateways, i.e., AND-split and AND-merge, whereas symbol **×** identifies a data-based exclusive gateway (XOR-split and XOR-merge), i.e., a point in the process where a condition must be evaluated in order to choose one path out of more. Finally, an encircled pentagon denotes an event-based gateway, i.e., a routing point in the process flow where event occurrence determines which is the path to follow. In this case, when a process path is chosen, all the others are discarded.

2.2. Temporal aspects in BPMN processes

Although business processes describe sequences of actions that follow an ordering that is homogeneous to the flowing of time, support of temporal perspective of business processes remains limited [2, 8, 16]. Among several design levels, conceptual modeling is probably the one being mostly affected by this lack of expressiveness [5], as there exist control and scheduling tools that manage certain temporal constraints during process run-time.

When modeling business processes from a temporal standpoint the temporal dimension of the process elements, whenever defined, depends on the modeling language used and on its semantics. Since many process modeling languages focus on control flow aspects, designing and visualizing temporal properties in process models becomes quite challenging, especially because of the lack of a common, clear temporal semantics.

Usually, it is assumed that the process sequence flow, gateways, and event triggering are instantaneous, that is, their execution does not consume any time or it consumes a fixed amount of time, which does not change during process execution [23]. This allows designers to ignore the temporal contribute brought by these constructs whenever computing the overall process duration or inferring temporal dependencies among flow elements.

An exception is represented by *catching events* that, once enabled, may need to wait for a certain amount of time prior to being activated by a trigger [13, 24]. For instance, a *message event* could potentially wait for one second or for an indefinite amount of time for the corresponding message trigger to arrive. However, it is reasonable to think that events will be triggered after a finite and practical amount of time and that any two event instances cannot occur exactly at the same point in time, assuming sufficient clock precision [23].

Activities take time to be executed. In general, a *duration* is defined as "an amount of time with known length, but no specific starting or ending instants" [25]. For example, a duration of "one week" is known to last seven days, but it can refer to any temporal block of seven consecutive days. It is widely accepted that duration is non-directional with respect to a timeline, that is, it is always positive. In this paper, for simplicity, we deal with a discrete time domain, i.e., we assume that time is discrete and isomorphic to natural numbers. As a consequence, the shortest interval has a duration of one time unit.

An activity, being either an atomic task or a compound subprocess, is expected to last a precise amount of time, within a range delimited by minimum duration MIN and maximum duration MAX values ($d \in [\text{MIN}, \text{MAX}]$). *Minimum duration* specifies that a certain activity must not complete earlier than a preset time point, that is, it should take at least a specified amount of time to be performed. A suitable clinical example for describing a minimum duration constraint is antibiotic therapy, which would result ineffective if administered for less than a certain number of days. On the other hand, a *maximum duration* constraint is used to set the upper-most time limit after which the activity is intended to have terminated. For example, the maximum duration for taking the database systems exam is fixed to 3 hours. Students that do not hand in their tests within 3 hours will not be evaluated.

Activities are associated to a life-cycle that determines their execution semantics. In this paper, we refer to the life-cycle depicted in Figure 1, adapted from [6, 13, 23]. When a process is instantiated, all the activities are in state *Inactive*. An activity is *enabled* and becomes *Ready* for execution when the

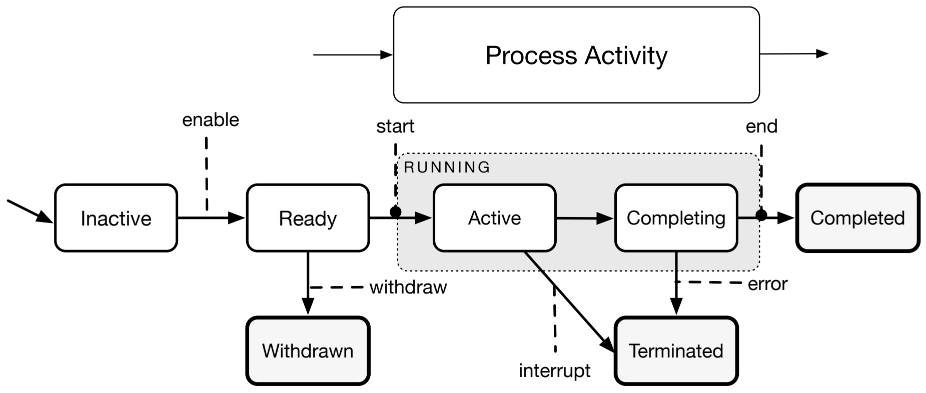


Figure 1: State diagram describing the life-cycle of a BPMN activity, adapted from [6, 13, 23].

number of tokens is available to activate it. When data and allocated resources are available, the activity changes from *Ready* to *Active*, through transition *start*. The start of the activity actually corresponds to an event that determines its beginning [13, 23]. When the activity ends without anomalies, it enters state *Completing*, which captures processing steps prior to activity completion, such as the end of process flows originated from non-interrupting events attached to its border. Then, the activity switches to state *Completed* through transition *end*. An activity moves from state *Ready* to *Withdrawn*, whenever it is placed on a process branch in the configuration of an event-based gateway that is not chosen during execution. Finally, any active or completing activity can switch to state *Terminated* in case of an execution *error*.

In this paper, when speaking about activity duration, we refer to the amount of time during which the activity is *RUNNING*, i.e., we do not consider the time span during which the activity is enabled but not yet started [3].

2.3. Well-structured process design

In general, being a graph-oriented process definition language, BPMN allows combining flow objects almost arbitrarily. However, this often leads to the definition of hardly-readable and complex process models, often containing semantic errors difficult to be detected during early process development phases [26]. In this setting, structural restrictions are desirable for increasing process model readability and prevent the onset of undesired deadlocks at run-time [27].

Process models are considered *structurally sound*, when they begin with exactly one start event, terminate with exactly one end event, and every flow node lies on a path from the start to the end event [28].

A BPMN process is said to be *well-structured* if for every node with multiple outgoing edges, i.e., a split node, it has a corresponding node with multiple incoming edges, i.e., a join node, such that the set of nodes delimited by the split and the join nodes form a Single-Entry-Single-Exit (SESE) region, and these regions within the process are properly nested [20]. Well-structured processes may be used to enhance modularity and improve local reasoning on large and complex process models.

A well-structured process is guaranteed to be sound if it is live [29]. The liveness property derives from the Petri Net context [30], and states that a Petri net is live if for every reachable state M and every transition t, there is a state M_i reachable from M which enables t. Liveness ensures the absence of deadlocks during process execution. A more detailed discussion regarding Petri Nets properties can be found in Appendix A.

The soundness property has been defined in the context of workflow nets, i.e., Petri nets having some peculiar properties [30]. Petri nets having the following structural restrictions identify an interesting sub-class of Petri nets, called workflow nets. A Petri net is a workflow net if and only if (i) it has a distinct source place (i.e., a single place that is not the target of any arc) and (ii) a distinct sink place (i.e., a single place that is not the source of any arc), and (iii) all of its nodes lie on some path from the source place to the sink place.

When adapted to process models, soundness guarantees that: all tasks can participate in a process instance, each process instance eventually terminates, and when a process instance terminates there is exactly one token in the end event [6].

For the reasons discussed, in this paper we prefer to define process models that are both well-structured and structurally sound, when possible. In particular, the duration patterns we define satisfy these properties.

However, the presence of exception flows, especially those originating from non-interrupting events attached to the task whose duration is constrained by our duration pattern, compromises both the well-structuredness and structural soundness of the overall process, since BPMN recommends ending each exception flow originating from a boundary event with its own end event [13]. Thus, the exception flows in our process models do not satisfy these structural properties. However, we will exploit BPMN process structural properties as a mean for guaranteeing that the designed process models are flexible and modular, and that they can be easily extended by adding new structured process branches, regions or subprocesses.

3. Business Process Models for Specifying Activity Duration

In this section, we formalize foundational concepts and introduce the different kinds of duration constraints addressed in this paper with the help of tables that summarize their main features and report meaningful examples.

The formal definition of process model used in this paper encompasses selected relevant BPMN elements and is provided below. The semantics of the introduced elements is assumed to be the one defined by BPMN [13, 18].

Definition 3.1 (Process Model). A process model $m = (N, C, \alpha, \epsilon_{tr}, \epsilon_{ty}, \beta, \delta, \gamma_r, \gamma_{ty}, \mathcal{L})$ consists of a finite non-empty set of flow nodes N and a finite non-empty set C of directed control flow edges. The set $N = \{A \cup G \cup E\}$ of flow nodes consists of the disjoint sets A of activities, G of gateways, and E

of events. E consists of the disjoint union of start events E_{start}, intermediate events E_{int} including also the set of boundary events E_B (i.e., $E_B \subseteq E_{int}$), and end events E_{end} characterizing m, namely $E = \{E_{start} \cup E_{int} \cup E_{end}\}$. Control flow $C \subseteq N \times N$ defines a connection between elements of N. Functions α, ϵ_{tr}, ϵ_{ty}, β, δ, γ_r, γ_{ty} associate a type to the elements of N. Namely, $\alpha : A \rightarrow \{task, subprocess\}$ distinguishes activities into tasks and subprocesses. $\epsilon_{tr} : E \rightarrow \{throwing, catching\}$ distinguishes events in those that throw a trigger and those that catch a result. It always holds that for each $s \in E_{start}$ $\epsilon_{tr}(e) = catching$ and for each $e \in E_{end}$ $\epsilon_{tr}(e) = throwing$. Function ϵ_{ty}: E $\rightarrow$ $\{none, message, signal, error, timer, conditional, escalation, cancel, multiple,$ $parallel, multiple, terminate\}$ associates a type to each event of E. $\beta : E_B \rightarrow$ $\{interrupting, non\text{-}interrupting\}$ associates the interrupting behavior to boundary events. $\delta : A \rightarrow 2^{E_B}$ is a function that associates to each activity $a \in A$ a set of boundary events. $\gamma_r : G \rightarrow \{split, merge\}$ assigns a gating mechanism to each gateway of G. $\gamma_{ty} : G \rightarrow \{parallel, exclusive, event{-}based\}$ assigns a routing type to each gateway of G. Finally, $\mathcal{L} : N \rightarrow \ell$ is a function that assigns a label ℓ, represented as a string, to each flow node in N.

Let θ be one of functions α, ϵ_{tr}, ϵ_{ty}, β, γ_r, and γ_{ty}. For practicality, in the remainder of this paper, we write $\theta(\{n_1, \ldots, n_m\}) = val$, as a shortcut for $\forall n_i \in \{n_1, \ldots, n_m\}\ \theta(n_i) = val$. Moreover, strings returned by labeling function $\mathcal{L}$ are not enclosed in double quotes (e.g. "Task"), but we use a special font style to denote strings (e.g., Task) to simplify the notation without compromising the interpretation of the process model.

A process model m having a unique start event s, a unique end event e, and each node $n \in \{A \cup G \cup E_{int}\}$ lies on a path from s to e, is structurally sound [28]. Despite being a desirable property, we do not require end events to be always unique as some exception flows, such as those outgoing of non-interrupting events, shall have with their own end event [13]. Instead, we always assume to deal with process models having a unique start event.

When composing multiple process parts into a comprehensive one, elements such as flow nodes or control flow edges may be used to ensure that their flows are correctly connected and their event handling mechanisms are consistent. We refer to the collection of such process elements as *connecting kit.*

Definition 3.2 (Connecting Kit). A connecting kit $Ck = (N_k, C_k)$ consists of a finite set of flow nodes N_k and a finite non-empty set of control flow edges C_k representing a graph not necessarily connected.

As previously mentioned, the main aim of this paper is to capture different kinds of duration constraints by means of dedicated process models, which we refer to as *duration patterns.* As outlined in Figure 2, starting from a process task (or region) to constrain, we design a duration pattern that captures the specified duration constraint. The duration pattern is attached to the activity through a set of additional BPMN flow and connecting elements, forming a connecting kit that is used to anchor the pattern to the task (or region) and to refine event handling. The resulting combination of the temporally constrained task (or region) with the duration pattern realized by the connecting kit, is a complete *duration-aware process model.*

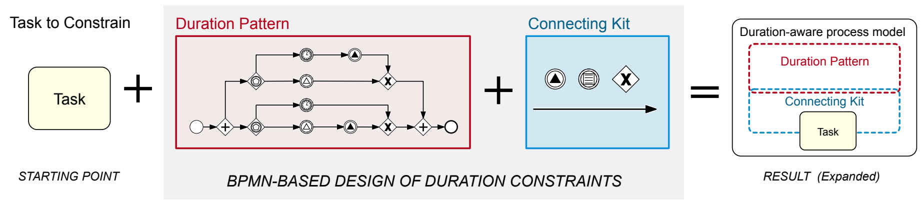


Figure 2: Starting from a process *Task* (or region) whose duration needs to be constrained, we design a (set of) *Duration Pattern*(s) that is attached to the task/region through suitable BPMN elements of a *Connecting Kit* to obtain a complete *Duration-aware process model.*

Such duration-aware process models may include different duration patterns depending on the kind of duration constraint that is represented.

In the remainder of the section, tables summarize and exemplify the different kinds of duration constraints addressed in this paper.

Table 1 describes duration constraints for process activities, considering duration constraints (i) for both upper and lower bounds; (ii) for either the lower or the upper bound; (iii) for disjoint ranges, i.e., less than the lower bound or greater than the upper bound. Table 2 introduces simple duration for tasks having attached boundary events. Table 3 considers the specification of simple duration for more complex process regions, such as SESE or non-SESE regions. Table 4 exemplifies some settings that require one to dynamically choose a proper duration range for an activity, after its initiation (deferred duration). Finally, Table 5 introduces how to specify a shifted duration for an activity, considering also resetting the duration count whenever environmental conditions change.

4. Specifying Simple Duration Constraints

In this section, we recall and formalize the structure of the duration-aware process model formerly proposed in [11] for specifying the simple duration of a process activity. This initial solution is the basic building block for the modeling of more complex constraints, addressed later in this work.

The simple duration of an activity can be specified through a well-structured and structurally sound process model, namely duration pattern ϕ_{simple}, that is meant to be suitably combined with the activity through connecting kit Ck_1.

Formally, $\phi_{simple} = (N, C, \alpha,\ \epsilon_{tr},\ \epsilon_{ty},\ \beta,\ \delta,\ \gamma_r, \gamma_{ty},\ \mathcal{L})$ has the following structure.

– $N = \{A \cup G \cup E\}$ is the set of flow nodes, where:
  $A = \varnothing$; $G = \{g_1,\ g_2,\ g_3,\ g_4,\ g_5,\ g_6\}$; $E = \{E_{start} \cup E_{int} \cup E_{end}\}$ where $E_{start} = \{s\}$, $E_{int} = \{e_1,\ e_2,\ e_3,\ e_4,\ e_5, e_6\}$, and $E_{end} = \{e\}$;

Simple Duration of an Activity (Task or Sub-Process) **Sec.** 4

Formal Notation

`SimpleDur`(ActivityName, MIN, MAX)

Duration Pattern and Connecting Kit

Basic pattern: ϕ_{simple}; Connecting kit: Ck_1

Constraint Description

The time span during which an activity is executed is restricted by minimum [MIN] and maximum [MAX] duration bounds.

Real-World Examples

- Preparation for the anesthesia in Caesarean section ranges between 8 to 41 minutes. [MIN and MAX]

One-Side Duration of an Activity (Task or Sub-Process) **Sec.** 4.1

Formal Notation

`MinSimpleDur`(ActivityName, MIN), `MaxSimpleDur`(ActivityName, MAX)

Duration Pattern and Connecting Kit

Basic pattern: variant of ϕ_{simple}; Connecting kit: Ck_1

Constraint Description

The time span during which an activity is executed is restricted by minimum [MIN] or maximum [MAX] duration bounds.

Real-World Examples

- The minimum acceptable duration of anticoagulation therapy for venous thromboembolism is at least 3 months. [MIN]
- Nimesulide should not be given for periods longer than 7 days in the treatment of acute pain. [MAX]

Out-Side Duration of an Activity (Task or Sub-Process) **Sec.** 4.2

Formal Notation

`OutSimpleDur`(ActivityName, MIN, MAX)

Duration Pattern and Connecting Kit

Basic pattern: variant of ϕ_{simple}; Connecting kit: Ck_1

Constraint Description

The time span during which an activity is executed is restricted by minimum [MIN] and maximum [MAX] duration bounds. Duration must be less than the minimum or greater than the maximum.

Real-World Examples

- The physical therapy may be applied in a intensive way, and in this case it cannot last more than 3 weeks. Otherwise, the same physical therapy may be applied in a standard way, but in this case it has to last more than 2 months to be effective. [MIN and MAX]

Table 1: Simple, one-side, and outside duration of activities.

SIMPLE DURATION OF AN ACTIVITY WITH BOUNDARY EVENTS **Sec.** 4.3
FORMAL NOTATION SimpleDurBE(ActivityName, MIN, MAX, EI, EN, EI_Handler, EN_Handler)
DURATION PATTERN AND CONNECTING KIT Basic pattern: ϕ_{simple}; Connecting kit: Ck_2
CONSTRAINT DESCRIPTION The duration of an activity with boundary interrupting [EI] or non-interrupting [EN] events is restricted by minimum [MIN] or maximum [MAX] duration bounds.
REAL-WORLD EXAMPLES • A cardiac Magnetic Resonance Imaging (MRI) scan usually lasts from 20 to 45 minutes, but must be interrupted if the patient has a panic attack or severe emotional distress. [MIN and MAX, EI] • Aspirin therapy for prevention of thrombotic events should last 28 days. If the patient experiences any upper gastrointestinal complication, gastro-protective co-therapy may be initiated. [MIN and MAX, EN]

Table 2: Simple duration of activities with boundary events.

- $C = \{(s, g_1), (g_1, g_2), (g_1, g_3), (g_2, e_1), (g_2, e_2), (g_3, e_3), (g_3, e_4), (e_4, e_6), (e_1, e_5), (e_5, g_4), (e_2, g_4), (e_3, g_5), (e_4, g_5), (g_5, g_6), (g_4, g_6), (g_6, e)\}$ is the set of control flow edges;
- $\alpha = \varnothing$;
- $\epsilon_{tr}(\{s, e_1, e_2, e_3, e_4\}) = catching$, $\epsilon_{tr}(\{e_5, e_6, e\}) = throwing$;
- $\epsilon_{ty}(\{s, e\}) = none$, $\epsilon_{ty}(\{e_1, e_3\}) = timer$, $\epsilon_{ty}(\{e_2, e_4, e_5, e_6\}) = signal$;
- $\beta = \varnothing$;
- $\delta = \varnothing$;
- $\gamma_r(\{g_1, g_2, g_3\}) = split$, $\gamma_r(\{g_4, g_5, g_6\}) = merge$;
- $\gamma_{ty}(\{g_1, g_6\}) = parallel$, $\gamma_{ty}(\{g_4, g_5\}) = exclusive$, and $\gamma_{ty}(\{g_2, g_3\}) = event\text{–}based$;
- $\mathcal{L}(s)$ = S, $\mathcal{L}(e_1)$ = *null*, $\mathcal{L}(e_2)$ = c_EXITED, $\mathcal{L}(e_3)$ = *null*, $\mathcal{L}(e_4)$ = c_EXITED, $\mathcal{L}(e_5)$ = t_maxViolated, $\mathcal{L}(e_6)$ = t_minViolated, $\mathcal{L}(e)$ = E, $\mathcal{L}(g_1)$ = G1, $\mathcal{L}(g_2)$ = G2, $\mathcal{L}(g_3)$ = G3, $\mathcal{L}(g_4)$ = G4, $\mathcal{L}(g_5)$ = G5, $\mathcal{L}(g_6)$ = G6.

Functions α, β, and δ are empty since $A = \varnothing$ and $E_B = \varnothing$. Events e_2 and e_4 are given the same label to highlight that they have the same triggering mechanism, i.e., they catch the same trigger.

Connecting kit $Ck_1 = (N_{k_1}, C_{k_1})$, where $N_{k_1} = \{e_1^k\}$ and $C_{k_1} = \varnothing$. Event e_1^k is an intermediate throwing signal event that triggers both events e_2 and e_4 of ϕ_{simple}, whenever they are actively listening (indeed, $\mathcal{L}(e_2) = \mathcal{L}(e_4)$).

The formal construct for specifying that the duration of an activity named ActivityName has to be between MIN and MAX time units is defined as

SimpleDur(ActivityName, MIN, MAX)

It generates a (sub)process model by using ϕ_{simple} and Ck_1 as follows:

Simple Duration of a SESE region	**Sec.** 5.1
Formal Notation	
SimpleDurSESE(EN, EX, MIN, MAX)	
Duration Pattern and Connecting Kit	
Basic pattern: ϕ_{simple}; Connecting kit: Ck_1	
Constraint Description	
The time span during which a sequential [$\rightarrow$], parallel [+], or exclusive [×] SESE region is executed is limited by minimum [MIN] and maximum [MAX] duration bounds.	
Real-World Examples	
• Exercise stress test lasts at least 45 minutes, comprehensive of preparation and monitoring times. [$\rightarrow$, MIN]	
• Therapy for H. Pylori eradication combines antibiotics and proton pump inhibitors for an overall duration of 7-10 days. [MIN and MAX, +]	
Simple Duration of a non-SESE region	**Sec.** 5.2
Formal Notation	
SimpleDurNonSESE(ENset, EXset, MIN, MAX)	
Duration Pattern and Connecting Kit	
Basic pattern: ϕ_{nSESE}, $\phi_{nSESEPred}$, $\phi_{nSESEGen}$; Connecting kit: Ck_3, Ck_4	
Constraint Description	
The time span during which a non-SESE region is executed is restricted by minimum [MIN] and maximum [MAX] duration bounds.	
Real-World Examples	
• On-scene time for adults trauma should be less than 10 minutes, during which airway with cervical spine control, breathing, circulation and disability are checked, depending on the patient's gravity and the probability of head trauma [MAX].	

Table 3: Simple duration of process regions.

SimpleDur(ActivityName, MIN, MAX) $= (N \cup N_{k_1} \cup \{a\}, C \cup \{(g_1, a), (a, e_1^k), (e_1^k, g_6)\}, \alpha \cup \{\{a \mapsto \textit{task}\}$ or $\{a \mapsto \textit{subprocess}\}\}, \epsilon_{tr} \cup \{e_1^k \mapsto \textit{throwing}\}, \epsilon_{ty} \cup \{e_1^k \mapsto \textit{signal}\}, \beta, \delta, \gamma_r, \gamma_{ty}, \mathcal{L}^*)$.

The new labeling function $\mathcal{L}^*$ extends $\mathcal{L}$ for e_1, e_4, e_1^k, and a, where $\mathcal{L}^*(e_1) =$ MAX, $\mathcal{L}^*(e_4) =$ MIN, $\mathcal{L}^*(e_1^k) =$ t_EXITED, and $\mathcal{L}^*(a) =$ ActivityName.

As outlined in Figure 2, the combination of Ck_1 and ϕ_{simple} with a specified activity results in a complete duration-aware process model returned by SimpleDur(Task, MIN, MAX).

Such process model can be included within more complex parent processes and represented as a collapsed subprocess. In other words, a duration-aware process model can be considered as a subprocess template that designers can use to specify task duration [11].

Without loss of generality, in Figure 3 the considered activity is a task, i.e., $\alpha(a) = \textit{task}$ and $\mathcal{L}^*(a) =$ Task.

The designed process allows us to manage three possible Task execution

Deferred Duration of an Activity	**Sec.** 6
Formal Notation DeferredDur(ActivityName, MIN1, MAX1, MIN2, MAX2, C1, C2) Duration Pattern and Connecting Kit Basic pattern: $\phi_{deferred}$; Connecting kit: Ck_1 Constraint Description An activity can be associated to two duration ranges $[min_1, max_1]$ and $[min_2, max_2]$, where a default one can be chosen prior to activity initiation. At run-time only one range is selected, based on how activity execution evolves. Therefore, the choice of which duration range applies is *deferred* with respect to activity initiation. Real-World Examples • Let us consider the treatment for pharyngitis. If the infection is viral, antibiotic therapy lasts 5 days, whereas for bacterial pharyngitis it should last 10 days. As it is hard to distinguish viral and bacterial causes based on symptoms alone, an empiric treatment is usually initiated, while a throat culture is grown. When results are obtained, treatment is specialized and continued based on the discovered nature of the infection. • Hospitalization following a diagnosis of appendicitis can last 1-2 days if surgery was routine, or up to 4 days if the removed appendix is found to be ruptured during the intervention.	

Table 4: Deferred duration of an activity: a duration range is chosen while the activity is being executed.

behaviors with respect to a given duration range: (i) Task is completed within the allowed duration, thus it lasts longer than the minimum desired time limit but ends before the maximum one, (ii) Task ends before the minimum time expected for its completion, thus violating its minimum duration constraint, (iii) Task is still executing when the maximum time limit allowed for completion is reached, thus raising a maximum duration violation.

The process begins when its start event S is triggered. The token arrives at parallel gateway G1, and the flow is split into three branches: two of them are directed to event-based gateways G2 and G3, while the last one is directed to Task. Event-based gateway G2 represents a branching point in the process where only one path is chosen (either flow1 or flow2) depending on which one of the events in its configuration is triggered first, i.e., event MAX on flow1, or event c_EXITED on flow2. Event-based gateway G3 has the same behavior as G2 but with respect to events MIN on flow3 and c_EXITED on flow4. When Task is completed, the token reaches the following (throwing) signal event t_EXITED, which is broadcast to be caught by corresponding (catching) signal events c_EXITED on flow2 and flow4.

(i) If Task completes within the defined duration range, signal event t_EXITED is caught only by signal event c_EXITED on flow2, since timer event MIN was triggered previously and, thus, flow4 was withdrawn.

(ii) If Task completes earlier than the minimum duration allowed signal event t_EXITED is caught by both events c_EXITED on flow2 and flow4. In this

Shifted Duration of an Activity	**Sec.** 7
Formal Notation ShiftedDur(ActivityName, MIN, MAX, ME, MAX_START, AltActivity) Duration Pattern and Connecting Kit Basic pattern: $\phi_{shifted}$; Connecting kit: Ck_1 Constraint Description The duration of a task is restricted by minimum [MIN] or maximum [MAX] duration bounds, and it is measured starting from a specific (*shifted*) moment indicating that a certain property begins to hold. We call this particular moment *milestone event*. Real-World Examples • Effective antibiotic therapy for endocarditis should last between 2 and 6 weeks, which are counted starting from the first day of negative cultures. [MIN and MAX]	
Shifted Duration of an Activity with Reset	**Sec.** 7
Formal Notation ShiftedDurR(ActivityName, MIN, MAX, ME, MAX_START, AltActivity, R) Duration Pattern and Connecting Kit Basic pattern: $\phi_{shiftRes}$; Connecting kit: Ck_1 Constraint Description Shifted duration count is reset [R] whenever this property stops holding earlier than the set minimum duration. Real-World Examples • Hospitalization must last between 24 and 36 hours, starting from the moment the patient defervesces (milestone event). If the patient has fever prior to 24 hours, duration count is reset and will re-start once fever disappears again. [MIN and MAX, R]	

Table 5: Shifted duration of an activity, considering also the possibility of resetting the duration count.

case, signal event t_minViolated on flow4 is broadcast to indicate that minimum duration has not been observed.

(iii) If Task is still executing when the maximum duration time limit allowed is reached, signal event t_EXITED is never caught, as both branches flow2 and flow4 are withdrawn (i.e., both timer events MIN and MAX are fired). Then, signal t_maxViolated located on flow2 is triggered to acknowledge that the maximum duration has been violated.

Signal events t_minViolated and t_maxViolated are used to detect violations and to trigger other processes designed to handle them, as explained in Section 8.

In dealing with task duration specification, we allow designers to use the proposed BPMN duration-aware process models as a ready-to-use building block whose timer events may be properly tuned to represent a specific duration.

For example, Figure 4 shows a process model including subprocess Duration-aware Antibiotic Therapy, generated by SimpleDur(Antibiotic Therapy, 7, 10), that represents task Antibiotic Therapy combined with its duration pattern.

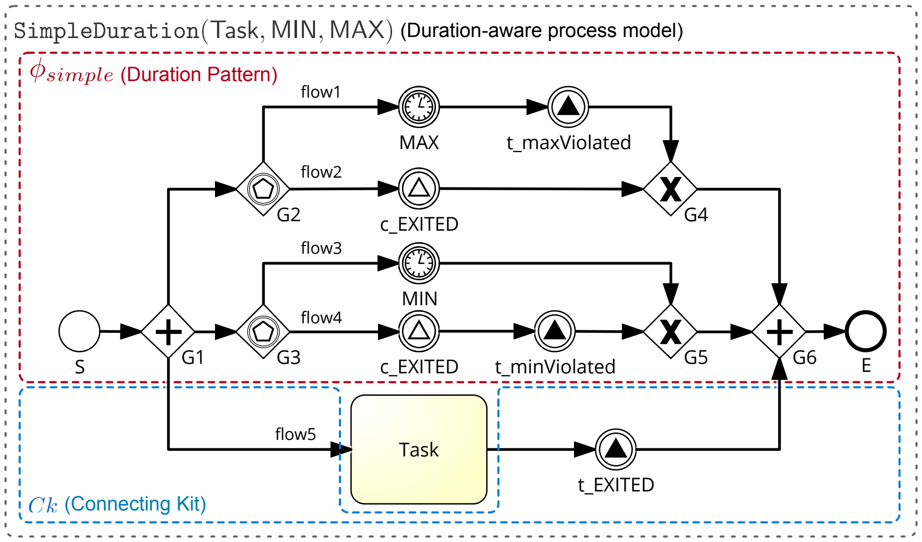


Figure 3: BPMN duration-aware process model specifying that the simple duration of Task is between MIN and MAX.

To make the proposed approach easier to use, we allow designers to also represent the whole duration-aware subprocess as a simple task characterized by a specialized icon such as a "clock", as done in [5, 14]. In particular, we propose also a new kind of task decorated with a clock flanked by $[MIN, MAX]$ for representing the combination of the considered task and the related duration pattern realized through a connecting kit. For the sake of readability, the considered time unit is explicitly specified only in the BPMN process models referring to real-world examples, while it is left implicit in the formal notation.

The bottom part of the Figure 4 shows the new task type Antibiotic Therapy for expressing a simple duration of $[7, 10]$ days. In this case, the label of the task is only Antibiotic therapy, since the notion of "duration-aware" is already represented by both the iconized clock and temporal range.

Anyhow, being conceptually and semantically defined through BPMN, the behavior of the underlying duration pattern remains clear, regardless of the chosen graphical representation.

4.1. Specifying the duration of an activity by constraining only one bound

When specifying a duration constraint, it could be useful to focus on either only the minimum or only the maximum duration. For these situations, we considered two different possibilities.

The first one is using the duration-aware process model, described above and showed in Figure 3, by specifying only the considered value to restrict. As an example, by specifying a duration range as [MIN,∞] we constrain the task to last at least MIN. Otherwise, by specifying a duration range as [0,MAX] we constrain the task to last at most MAX.

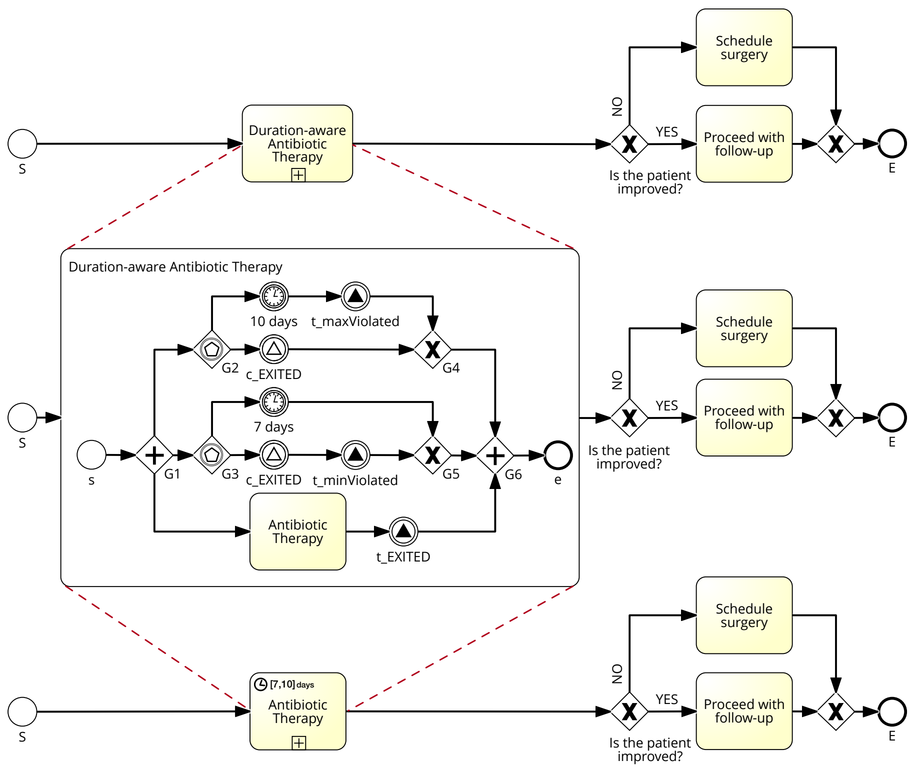


Figure 4: Process model Duration-aware Antibiotic Therapy obtained through `SimpleDur`(Antibiotic Therapy, 7, 10) and depicted (top) as subprocess included in a more complex parent process and (bottom) as a new task type including a clock icon.

The second possibility is the definition of the following *One-Side Simple Duration Patterns*, where we consider only the specification of either the minimum duration or the maximum duration of a task. In Figure 5, we report the designed BPMN duration-aware process model for constraining the minimum duration of Task, through the formal construct `MinSimpleDur`(ActivityName, MIN). For example, this kind of pattern can be used for specifying that haemodialysis must last at least 3 hours.

On the other hand, in Figure 6 we report the designed BPMN duration-aware process model, obtained through `MaxSimpleDur`(ActivityName, MAX), for constraining the maximum duration of Task. For example, this pattern can be used for specifying that Nimesulide should not be administered for longer 7 days in the treatment of acute pain (cf. Table 1).

The formal construct for the two introduced scenarios stems directly from the previously presented formalization of simple duration. Indeed, duration pattern ϕ_{simple} is simplified to consider the process fragment containing the specification of either the maximum or the minimum allowed duration, respectively.

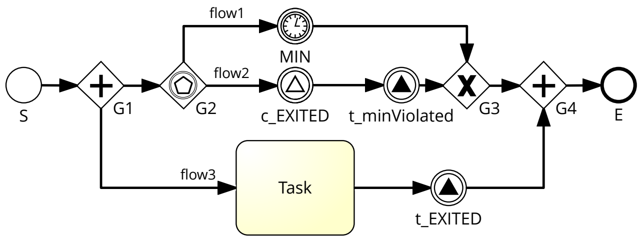


Figure 5: BPMN duration-aware process model for specifying the minimum duration of Task.

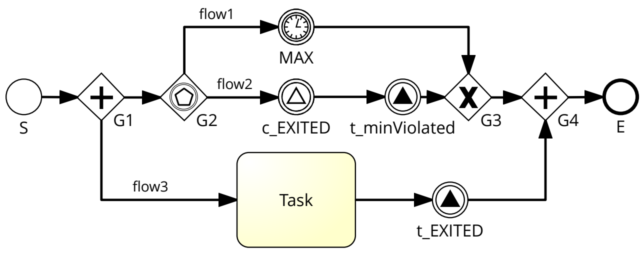


Figure 6: BPMN duration-aware process model for specifying the maximum duration of Task.

4.2. Specifying the duration of an activity outside of a certain range

In some application cases, activities must last for an amount of time that is either less than a certain value (called MIN) or more than a greater value (called MAX), that is, any duration between MIN and MAX is disallowed.

For example, the rehabilitation of a patient requires different levels of treatment, each one administered for a certain amount of time. Restoring functional ability and quality of life requires the administration of either an intensive treatment, which must end within a given time span (MIN), or of a long-term treatment, that must be given for a time span longer than a specified one (MAX).

The process model depicted in Figure 7 shows the duration-aware pattern for managing this setting. This case needs to deal with two alternative options: either Task completes before timer event MIN is triggered, or it must complete only after an amount of time MAX has passed.

Since the two cases are alternative, but not independent, as "completing after MIN" may include also the case "completing after MAX", the process elements that compose the proposed duration pattern are nested. That is, compared to duration pattern ϕ_{simple}, the process branches modeling minimum duration, i.e., flow1 and flow2 include also the SESE region delimited by G3 and G4, capturing the upper duration bound. The timer event in the context of G3 measures an amount of time equal to MAX-MIN since the SESE region delimited by G3 and G4 is nested and, thus, not enacted in parallel with Task.

The formal construct `OutSimpleDur`(ActivityName, MIN, MAX) may be spec-

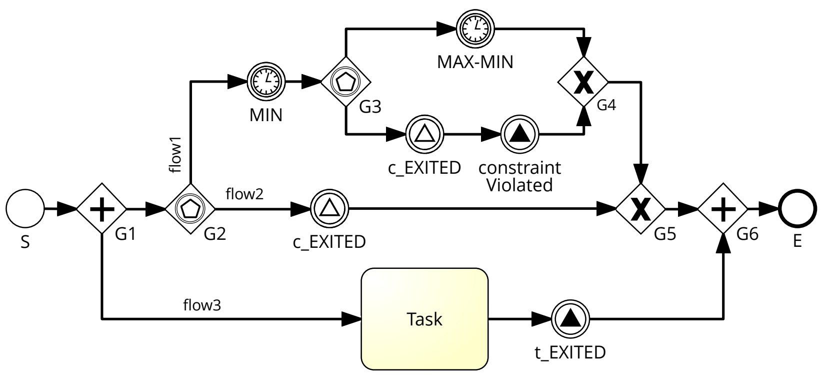


Figure 7: BPMN duration-aware process model for specifying that the duration of Task sould be less than MIN or greater than MAX.

ified by suitably adapting pattern ϕ_{simple}.

4.3. Specifying the duration of an activity with boundary events

In order to provide a complete picture of activity simple duration specification, we considered the possibility of having boundary events influencing activity execution and potentially affecting its overall duration.

When placed on the boundary of a task or subprocess, an intermediate event is used to represent exception or compensation handling [13]. In accordance with the BPMN standard [13], in this paper we refer to the (sub)processes that handle boundary events or originate from them to as "event handlers".

Once the event occurrence is consumed, the activity can be interrupted if the event is an interrupting one, or activity execution can continue in parallel with the event handler, if the event is non-interrupting.

Although multiple interrupting boundary events can be attached to the same activity, only one of such handlers can be executed at a time, for obvious reasons. Conversely, an unlimited number of non-interrupting event handlers can be modeled and executed in parallel, while the activity continues its execution.

In this latter setting, it is important to recall the previously explained semantic states of a process activity, shown in Figure 1. In particular, a BPMN activity can switch to state *Completed* only when all the non-interrupting event handlers attached to its boundary are completed. Indeed, an activity moves from state *Active* to state *Completing* when its execution has finished, but the attached non-interrupting boundary handlers are still running.

The previously introduced formal construct SimpleDur(ActivityName, MIN, MAX) can be revisited by considering the addition of both interrupting and non-interrupting events attached to the boundary of the considered activity.

In detail, we consider having an activity with two intermediate boundary events EI and EN attached to its border, where EI is interrupting and EN is non-interrupting.

Whereas the core duration pattern is again ϕ_{simple}, the connecting kit must be redefined to capture and synchronize also the exception flows originating from the boundary events. In detail, connecting kit $Ck_2 = (Nk_2, Ck_2)$, where $Nk_2 = \{e_1^k, e_2^k, e_3^k, g_1^k\}$ and $Ck_2 = \varnothing$.

The formal construct for specifying that the duration of an activity named ActivityName with attached boundary events EI and EN (and respective event handlers EI_Handler and EN_Handler) has to be between MIN and MAX time units is defined as

`SimpleDurBE`(ActivityName, MIN, MAX, EI, EN, EI_Handler, EN_Handler)

It generates a (sub)process model by using ϕ_{simple} and Ck_2 as follows:

`SimpleDurBE`(ActivityName, MIN, MAX, EI, EN, EI_Handler, EN_Handler) $= (N \cup N_{k_2} \cup \{a\},\ C \cup Ck_2 \cup \{(g_1, a),\ (a, e_1^k),\ (e_1^k, g_1^k),\ (ei, e_2^k),\ (e_2^k, a_2),\ (a_2, g_1^k),\ (g_1^k, g_6), (en, a_1), (a_1, e_3^k)\}, \alpha \cup \{\{a \mapsto task\}$ or $\{a \mapsto subprocess\}\} \cup \{\{a_1, a_2\} \mapsto subprocess\},\ \epsilon_{tr} \cup \{\{e_1^k, e_2^k, e_3^k\} \mapsto throwing\} \cup \{\{ei, en\} \mapsto catching\},\ \epsilon_{ty} \cup \{\{e_1^k, e_2^k\} \mapsto signal\} \cup \{e_3^k \mapsto none\} \cup \{e_i \mapsto t_1\} \cup \{e_n \mapsto t_2\}$ where $t_1, t_2 \in \{signal, message, timer, conditional, escalation\},\ \beta \cup \{ei \mapsto interrupting\} \cup \{en \mapsto non{-}interrupting\},\ \delta \cup \{a \mapsto \{ei, en\}\},\ \gamma_r \cup \{g_1^k \mapsto merge\},\ \gamma_{ty} \cup \{g_1^k \mapsto exclusive\},\ \mathcal{L}^*)$.

The new labeling function $\mathcal{L}^*$ extends $\mathcal{L}$ for e_1, e_4, e_1^k, e_2^k, e_3^k, ei, en, a, a_1, a_2 and g_1^k, where $\mathcal{L}^*(e_1) =$ MAX, $\mathcal{L}^*(e_4) =$ MIN, $\mathcal{L}^*(e_1^k) =$ t_EXITED, $\mathcal{L}^*(e_2^k) =$ t_EXITED, $\mathcal{L}^*(e_3^k) =$ E2, $\mathcal{L}^*(ei) =$ EI, $\mathcal{L}^*(a_2) =$ EI_Handler, $\mathcal{L}^*(en) =$ EN, $\mathcal{L}^*(a) =$ ActivityName, $\mathcal{L}^*(a_1) =$ EN_Handler, and $\mathcal{L}^*(g_1^k) =$ G7.

Figure 8 shows the duration-aware process model obtained through `Simple-DurBE`(Task, MIN, MAX InterruptingE, Non−InterruptingE, InterruptingE Handler, Non−InterruptingE Handler). Without loss of generality, the considered activity is represented as a task, i.e., $\alpha(a) = task$ and $\mathcal{L}^*(a) =$ Task.

As for the type of boundary events, in Figure 8 we employ a *none* intermediate event placed on the activity boundary although this kind of event is not defined in BPMN as a possible *catching* event. We chose to use it as a graphical expedient for the sake of simplicity, as the behavior of the process model generated by `SimpleDurBE`(ActivityName, MIN, MAX, EI, EN, EI_Handler, EN_Handler) remains the same for any of the following BPMN event triggers: *Signal*, *Timer*, *Message*, *Conditional*, and *Escalation*.

Specifically, any *Signal* or *Message* event can be sent by any external participant or process instance. A *Conditional* event is triggered when a condition becomes true, whereas an *Escalation* event triggers a reaction to a specific business situation. *Timer* events are assumed to be triggered at a fixed *timeDate*, such as "February 23rd 2015", *timeCycle* for example "every Monday at 9.00 a.m." or *timeDuration*, such as "2 hours" [13]. Of course, when defined with an absolute *timeDate*, their triggering depends on when the process is executed.

In the presented solution, the exception flow outgoing from interrupting event InterruptingE is alternative to the normal flow outgoing from Task. In this setting, compared to the process of Figure 3, the semantic state of Task changes to *Terminated* if the interrupting event is triggered. Therefore, signal

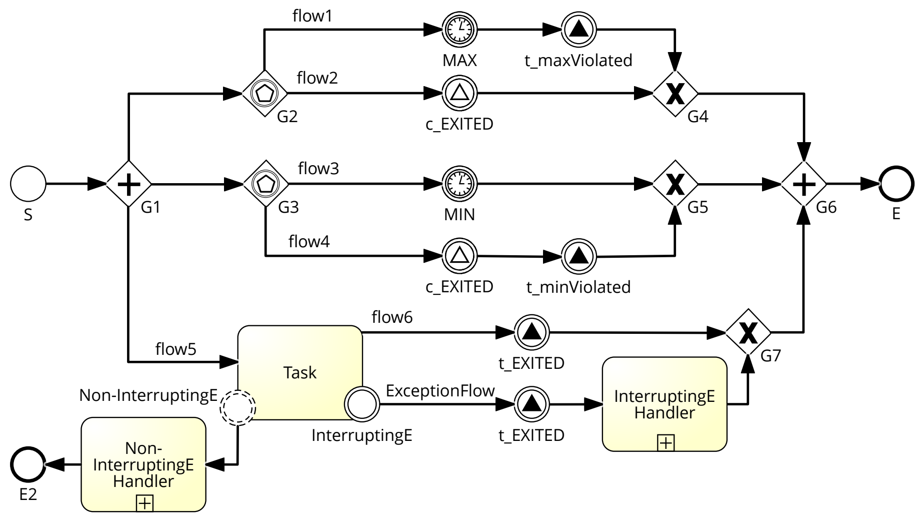


Figure 8: Duration-aware process model representing the duration of a Task having attached interrupting and non-interrupting intermediate boundary events InterruptingE and Non-InterruptingE.

event t_EXITED can be replicated within ExceptionFlow to be used in case of minimum constraint violation caused by early task interruption. Besides this, no substantial change in the model is required to detect constraint violation, as the behavior of ϕ_{simple} remains unvaried.

Regarding the occurrence of interrupting boundary events and timer events MIN and MAX, it is worth noticing that an interrupting event may occur either earlier or later than MIN, conjectured that it occurs before Task is completed. Conversely, an interrupting event can occur after the maximum duration only if Task is violating the constraint set by MAX. That said, we can conclude that interrupting boundary events can directly raise minimum constraint violations whenever Task is interrupted before the triggering of MIN. However, being designed as an exception, this latter kind of violation is a predictable temporal exception, properly captured by event handlers in the process model.

Construct `SimpleDurBE`(ActivityName, MIN, MAX, EI, EN, EI_Handler, EN_Handler) may be easily adapted to deal with multiple boundary events or to boundary events of only one kind (i.e., either all of kind interrupting or non-interrupting). When multiple boundary events are attached to the activity border, then process elements must be properly added to connecting kit Ck_2. For each interrupting event, a signal event (similar to e_2^k) and the edge connecting it to gateway g_1^k must be added to Ck_2. For each non-interrupting event, the connecting kit must include an end event (similar to e_3^k).

As for boundary events of only one kind, consider the process model of Figure 9, depicting activity cardiac Magnetic Resonance Imaging, which lasts 20–45 minutes.

Such process model is generated through `SimpleDurBE`(Magnetic Resonance Imaging, 20, 45, Patient is panicking, *null*, Handle patient panic, *null*). Regardless of duration constraints, magnetic resonance imaging must be interrupted any time the patient experiences a panic attack. This interruption is captured by boundary conditional event Patient is panicking, which is usually expected to occur during the first minutes of the exam. If task Magnetic Resonance Imaging needs to be interrupted, its minimum duration is violated. However, since the violation is predictable and modeled in the process, planned activities Calm patient down and Schedule MRI with sedation are enacted as part of event handler Handle patient panic.

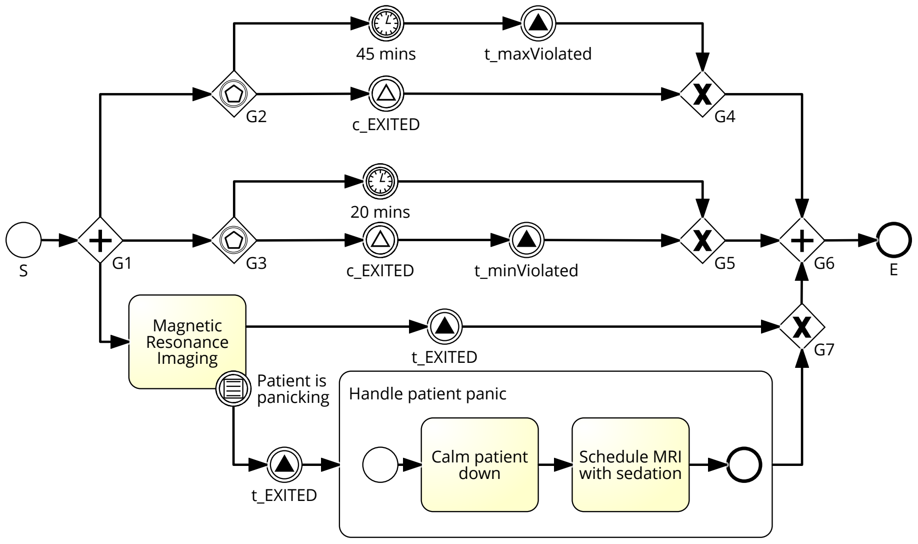


Figure 9: Process model representing minimum and maximum duration constraints of 20 and 45 minutes for Magnetic Resonance Imaging, which is interrupted anytime a Patient is panicking. Task interruption is handled by subprocess Handle patient panic.

Instead, non-interrupting boundary events do not directly affect the activity execution with respect to duration constraints. However, the execution of event handlers may have some repercussion on the overall activity duration.

As previously mentioned, for a BPMN activity to be in state *Completed*, all attached non-interrupting event handlers must have reached a completion state. With respect to activity duration, this translates into a relaxation of the control over the exact amount of time needed for Task to execute, since an activity that is completed within the expected duration range might have to wait for its non-interrupting event handlers to proceed. At the level of abstraction adopted in the process of Figure 8, there is no means to distinguish if the maximum duration constraint is violated by the activity itself, thus the activity is still in state *Active*, or by the non-interrupting event handlers started by Non-InterruptingE, i.e., the activity is in state *Completing*.

Additionally, if we consider that exception handlers may themselves be composed of activities, constraining the duration of an activity subjected to the occurrence of handlers executed in parallel becomes challenging. Similarly, we may have a case of an activity that completes earlier than its minimum expected duration, but it remains in state *Completing* while it waits for attached non-interrupting event handlers to complete. In this case, its overall duration appears to be within the desired time constraints. From the discussed scenario we can evince that, to model the real duration of an activity, any non-interrupting event handler associated to Task must observe the same maximum duration bound set for Task in order to avoid delaying activity completion.

5. Specifying Simple Duration Constraints of Process Regions

In this section, we discuss how to specify the simple duration of Single-Entry-Single-Exit (SESE) process regions composed solely of tasks or delimited by split and merge parallel or exclusive gateways. Then, we consider specifying the duration of non-Single-Entry-Sigle-Exit (non-SESE) regions and extend the set of process patterns presented in [11] to constrain the duration of arbitrarily selected regions having multiple entry or exit nodes.

5.1. Simple duration of Single-Entry-Single-Exit regions

Well-structuredness is a desirable property of process models, which promotes ease of comprehension and design errors reduction [20]. As explained in Section 2.3, structured process models are decomposable into Single-Entry-Single-Exit regions, SESE regions for short.

Formally, a Single-Entry-Single-Exit region $R_{(en,ex)} = (N_R, C_R)$, where N_R is a set of flow nodes and C_R is a set of control flow edges connecting the nodes of N_R, is a (subset of a) process model delimited by two flow nodes $en, ex \in N_R$ having the following properties [32]: (i) Every path from the start of the process to ex includes en; (ii) Every path from the end of the process to en includes ex; (iii) Every cycle containing en also contains ex and vice-versa.

Given a SESE region $R_{(en,ex)}$, we refer to flow node en as the "entry node" of the region and to ex as its "exit node".

Regardless of its internal structure, a SESE region $R_{(en,ex)}$ delimited by en and ex can be easily attached to duration pattern ϕ_{simple} through connecting kit Ck_1 defined in Section 4.

The formal construct for specifying that the duration of a SESE region delimited by entry node EN and exit node EX has to be between MIN and MAX time units is defined as

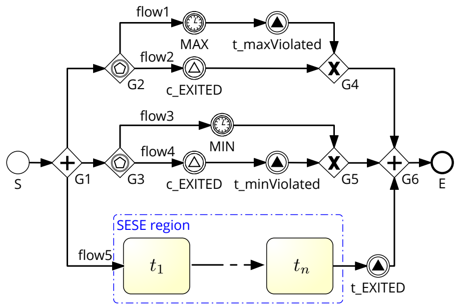
$$\texttt{SimpleDurSESE}(\mathsf{EN}, \mathsf{EX}, \mathsf{MIN}, \mathsf{MAX})$$

It generates a (sub)process model by using ϕ_{simple} and Ck_1 as follows:

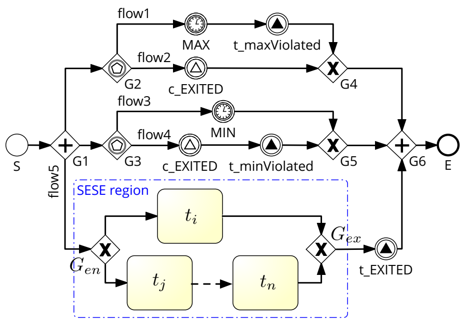
$\texttt{SimpleDurSESE}(\mathsf{EN}, \mathsf{EX}, \mathsf{MIN}, \mathsf{MAX}) = (N \cup N_R \cup N_{k_1}, C \cup C_R \cup \{(g_1, en),$ $(ex, e_1^k),\ (e_1^k, g_6)\},\ \alpha,\ \epsilon_{tr} \cup \{e_1^k \mapsto \mathit{throwing}\},\ \epsilon_{ty} \cup \{e_1^k \mapsto \mathit{signal}\},\ \beta,\ \delta,\ \gamma_r,$

γ_{ty}, $\mathcal{L}^*$). The new labeling function $\mathcal{L}^*$ extends $\mathcal{L}$ for e_1, e_4, and e_1^k, where $\mathcal{L}^*(e_1) = \mathsf{MAX}$, $\mathcal{L}^*(e_4) = \mathsf{MIN}$, and $\mathcal{L}^*(e_1^k) = \mathsf{t\_EXITED}$. Moreover, $\mathcal{L}^*$ assigns a label to every node of N_R (e.g., $\mathcal{L}^*(en) = \mathsf{EN}$, $\mathcal{L}^*(ex) = \mathsf{EX}$).

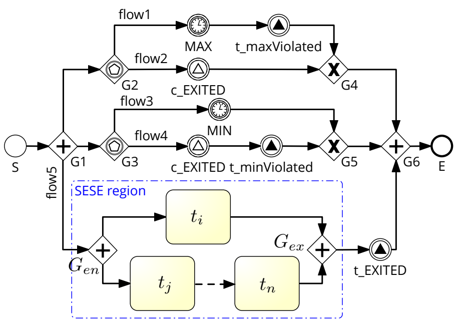
Figure 10 shows the duration-aware process models obtained by combining basic kinds of SESE regions with duration pattern ϕ_{simple} and connecting kit Ck_1.



(a) SESE region $R_{(t_1,t_n)}$ containing a sequence of tasks.



(b) SESE region $R_{(G_{en},G_{ex})}$ delimited by exclusive gateways.



(c) SESE region $R_{(G_{en},G_{ex})}$ delimited by parallel gateways.

Figure 10: Duration pattern ϕ_{simple} applied to different Single-Entry-Single-Exit (SESE) regions. (a) Sequence of tasks $t_1, \ldots t_n$; (b) SESE region delimited by exclusive gateways G_{en} and G_{ex}; and (c) SESE region delimited by parallel gateways G_{en} and G_{ex}.

Figure 10(a) shows the duration-aware process model designed to specify the duration of a SESE region composed by a sequence of tasks $t_1 \ldots t_n$ and obtained through $\mathtt{SimpleDurSESE}(\mathsf{t_1}, \mathsf{t_n}, \mathsf{MIN}, \mathsf{MAX})$. The simple duration of $R_{(\mathsf{t_1},\mathsf{t_n})}$ is the time that elapses from the beginning of the first element of the sequence $\mathsf{t_1}$, i.e., the entry node, to the ending of last one $\mathsf{t_n}$, i.e., the exit node. Signal event $e_1^k \in Ck_1$ (labeled as $\mathsf{t\_EXITED}$) is placed after the exit node $\mathsf{t_n}$. The design principles illustrated for the sequence of tasks may be generalized to any sequence of flow nodes other than gateways, i.e., activities and events.

Figure 10(b) shows the duration-aware process model obtained through $\mathtt{SimpleDurSESE}(\mathsf{G_{en}}, \mathsf{G_{ex}}, \mathsf{MIN}, \mathsf{MAX})$, which captures the simple duration of a region $R_{(\mathsf{G_{en}},\mathsf{G_{ex}})}$ delimited by exclusive gateways.

Similarly, the process model of Figure 10(c) considers duration of a SESE region delimited by parallel gateways $\mathsf{G_{en}}$ and $\mathsf{G_{ex}}$.

For readability, Figure 10(b–c) depicts only two process branches within exclusive and parallel blocks, but the discussed solutions hold for SESE regions enclosing an arbitrary number of internal flow branches.

In general, SESE regions can be trivial (e.g., a single activity) or arbitrarily complex (e.g., a whole well-structured process model), as they may contain other nested SESE regions. However, since the proposed solution exploits modularity, parallel gateway g_1 of ϕ_{simple} can always be connected to the entry node of the SESE region of interest, while the exit node of the region can be connected to signal event e_1^k of Ck_1 that leads to parallel gateway g_6 of ϕ_{simple}.

5.2. Simple duration of non-Single-Entry-Single-Exit regions

Compared to dealing with SESE regions, when considering non-Single-Entry-Single-Exit regions it is important to keep in mind that duration is influenced by multiple starting and ending points that need to be "synchronized" [11].

In addition, both the structure of the non-SESE region and the way it is connected to the remaining elements of a process model affect the structure of the related duration patterns.

For this reason, we start by dealing with basic non-SESE regions and consider an increasing number of structural features throughout this section, to show how duration patterns may be adapted to deal with complex non-SESE regions.

As a first step, we derive non-SESE regions from the basic kinds of SESE regions outlined in Figure 10. Trivially, there is no way of defining a non-SESE region spanning a sequence of tasks. Similarly, considering non-SESE regions spanning multiple alternative process branches is nonsense, since only one of them is executed at a time. Thus, we are interested in non-SESE regions that span at least two parallel process branches.

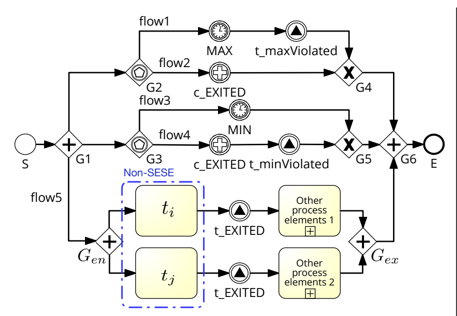
By taking the SESE region of Figure 10(c) as a reference, we retrieve four possible arrangements of activities enclosed within a SESE region $R_{(G_{en},G_{ex})}$ delimited by parallel gateways. We report them in Figure 11 and discuss how to specify the duration of the non-SESE region composed by such activities.

In Figure 11, process elements located within $R_{(G_{en},G_{ex})}$ but not belonging to the non-SESE region to constrain are represented as collapsed subprocesses labeled Other process elements. We refer to such process elements as *predecessors* or *successors*, based on their position with respect to the entry and exit nodes of the considered non-SESE region, framed by a (blue) dash and dotted line.

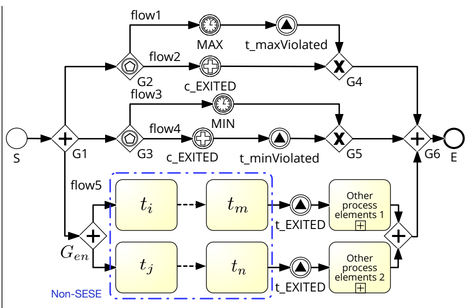
In particular, Figure 11 shows all the possible arrangements of predecessors with respect to the entry nodes of the considered non-SESE region.

The non-SESE region of Figure 11(a) is constituted by two tasks t_i and t_j that do not have predecessors, but both have successors. The entry and exit nodes of the non-SESE region coincide.

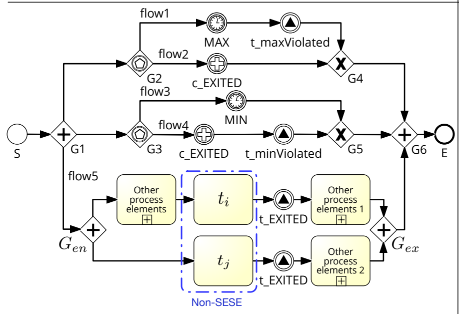
The non-SESE region of Figure 11(b) generalizes the one in Figure 11(a): it is constituted by groups of tasks $t_i \dots t_m$ and $t_j \dots t_n$ that do not have predecessors, but both have successors. Entry points are t_i and t_j, whereas exit points are t_m and t_n.



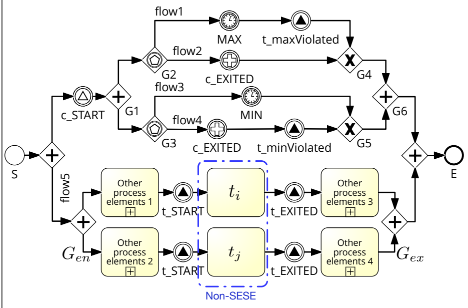
(a) Tasks t_i and t_j without predecessors w.r.t. t_i, t_j. Entry nodes: t_i, t_j; exit nodes: t_i, t_j. Minimal SESE region: $R_{(G_{en},G_{ex})}$.



(b) Group of tasks $t_i \dots t_m$ and $t_j \dots t_n$ with no predecessors w.r.t. t_i, t_j. Entry nodes: t_i, t_j; exit nodes: t_m, t_n. Minimal SESE region: $R_{(G_{en},G_{ex})}$.



(c) Tasks t_i and t_j with predecessors only w.r.t. t_i. Entry nodes: t_i, t_j; exit nodes: t_i, t_j. Minimal SESE region: $R_{(G_{en},G_{ex})}$.



(d) Tasks t_i and t_j with predecessors w.r.t. t_i and t_j. Entry nodes: t_i, t_j; exit nodes: t_i, t_j. Minimal SESE region: $R_{(G_{en},G_{ex})}$.

Figure 11: Non-Single-Entry-Single-Exit (Non-SESE) regions distributed across two flow branches of a SESE region delimited by parallel gateways. (a) Coinciding entry and exit points, no predecessors; (b) No predecessors; (c) There is at least one entry node without predecessors; and (d) All entry nodes have predecessors.

The non-SESE region of Figure 11(c) is composed by two tasks t_i and t_j where only the entry node t_i has predecessors. The entry and exit nodes of the non-SESE region coincide, but this case may be generalized to groups of tasks.

Finally, the non-SESE region of Figure 11(d) is composed by two tasks t_i and t_j that both have predecessors and successors. The entry and exit nodes of the non-SESE region coincide, but this case may be generalized to groups of tasks.

When dealing with non-SESE regions, in order to maintain the design well-structured, duration patterns must be connected to the entry and exit nodes of the *smallest SESE region* enclosing the non-SESE region of interest, e.g., $R_{(G_{en},G_{ex})}$ in Figure 11. However, this scenario requires that process elements preceding and succeeding the entry and exit nodes of the considered non-SESE region are ignored by the duration pattern and properly managed. Indeed, multiple exit nodes must be synchronized and the presence of predecessors influences when the duration pattern must be enacted.

Let us begin with considering only multiple exit nodes. If a non-SESE region has multiple exit nodes, they shall be synchronized, as the last one to complete determines the completion time of the whole region. Instead, since duration is calculated starting from the first among all entry nodes being enacted, entry nodes do not need to be synchronized.

Consider duration pattern ϕ_{simple}, introduced in Section 4 and used to specify the duration of SESE regions in Section 5.1. In Figure 10, signal events e_2 and e_4 of ϕ_{simple}, both labeled $\mathsf{c\_EXITED}$, are meant to catch the corresponding signal event $\mathsf{t\_EXITED}$ of connecting kit Ck_1, placed after the exit point of the considered region.

By following the same design principles, a throwing signal event $\mathsf{t\_EXITED}$ shall be placed after each one of the exit nodes of the non-SESE region to capture and synchronize multiple exit nodes. However, in order to achieve synchronization, events $\mathsf{c\_EXITED}$ must be of kind *parallel multiple*, that is, they are assigned arbitrary number of triggers and all of them are required for the event to fire [13]. In this way, events $\mathsf{c\_EXITED}$ located in the context of event-based gateways $\mathsf{G2}$ and $\mathsf{G3}$, respectively, are triggered only when all the associated $\mathsf{t\_EXITED}$ events placed on each exit edge of the non-SESE region have fired.

Minimum duration is violated when all events $\mathsf{t\_EXITED}$ are triggered earlier than MIN, whereas maximum duration is violated when at least one element of the region lasts longer than the maximum time allowed for completion MAX.

We call ϕ_{nSESE} this variant of duration pattern ϕ_{simple}, where the only difference is given by $\epsilon_{ty}(\{e_2, e_4\}) = \mathit{parallel\,multiple}$.

Let us consider a non-SESE region $NR_{(\{en_1,en_2,\dots,en_n\},\{ex_1,ex_2,\dots,ex_m\})} = (N_{NR}, C_{NR})$ delimited by a set of entry nodes $\mathsf{ENset} = \{en_1, en_2, \dots, en_n\} \in N_{NR}$ and a set of exit nodes $\mathsf{EXset} = \{ex_1, ex_2, \dots, ex_m\} \in N_{NR}$, where there exist a partial order between entry and exit nodes.

Let $R_{(en,ex)} = (N_R, C_R)$ be the minimal SESE region enclosing $NR_{(\mathsf{ENset},\mathsf{EXset})}$. The nodes and edges of $R_{(en,ex)}$ may always be partitioned into three sets: the set $P = \{en, p_1, ..., p_n\}$ of flow nodes preceding the elements of ENset, the set N_{NR}, and the set $S = \{s_1, ..., s_m, ex\}$ of flow nodes succeeding the elements of EXset, i.e., $N_R = \{P \cup N_{NR} \cup S\}$, where $|P| \geq 0$, $|N_{NR}| \geq 2$, and $|S| \geq 0$.

Without loss of generality, let us consider the case of a non-SESE region having two exit nodes, i.e., $|\mathsf{EXset}| = 2$. For this basic case, connecting kit $Ck_3 = \{N_{k_3}, C_{k_3}\}$, where $N_{k_3} = \{e_1^k, e_2^k\}$ and $C = \varnothing$.

The formal construct for specifying that the duration of a non-SESE region $NR_{(\mathsf{ENset},\mathsf{EXset})} = (N_{NR}, C_{NR})$ delimited by a list of entry nodes ENset and a list of exit nodes EXset with $|\mathsf{EXset}| = 2$, and enclosed within SESE region $R_{(en,ex)} = (N_R, C_R)$ has to be between MIN and MAX time units is defined as

$$\texttt{SimpleDurNonSESE}(\mathsf{ENset}, \mathsf{EXset}, \mathsf{MIN}, \mathsf{MAX})$$

It generates a (sub)process model by using ϕ_{nSESE} and Ck_3 as follows:

$\texttt{SimpleDurNonSESE}(\mathsf{ENset}, \mathsf{EXset}, \mathsf{MIN}, \mathsf{MAX}) = (N \cup N_R \cup N_{k_3},\ C \cup C_R \cup \{(g_1, en),\ (ex_1, e_1^k),\ (e_1^k, s_1),\ (ex_2, e_2^k),\ (e_2^k, s_2),\ (ex, g_6)\},\ \alpha,\ \epsilon_{tr} \cup \{\{e_1^k, e_2^k\} \mapsto \mathit{throwing}\},\ \epsilon_{ty} \cup \{\{e_1^k, e_2^k\} \mapsto \mathit{signal}\},\ \beta,\ \delta,\ \gamma_r,\ \gamma_{ty},\ \mathcal{L}^*)$.

The new labeling function $\mathcal{L}^*$ extends $\mathcal{L}$ for e_1, e_4, e_1^k, and e_2^k where $\mathcal{L}^*(e_1) = \mathsf{MAX}$, $\mathcal{L}^*(e_4) = \mathsf{MIN}$, $\mathcal{L}^*(e_1^k) = \mathsf{t\_EXITED}$, and $\mathcal{L}^*(e_2^k) = \mathsf{t\_EXITED}$. As in the previous case, $\mathcal{L}^*$ assigns a label to every node of N_R (e.g., $\mathcal{L}^*(en) = \mathsf{EN}$, $\mathcal{L}^*(ex) = \mathsf{EX}$).

Formal construct `SimpleDurNonSESE`(ENset, EXset, MIN, MAX) can be easily generalized for dealing with non-SESE regions having more than two exit nodes: it is sufficient to add one signal event e_i^k to connecting kit Ck_3 for each exit node ex_i of EXset, the edge (ex_i, e_i^k) connecting them, and the edge (e_i^k, s_i) connecting the signal event of the connecting kit with the first process successor encountered after ex_i.

Duration pattern ϕ_{nSESE} can be seen in the duration-aware process models of Figure 11(a)–(c), properly combined through connecting kit Ck_3 with the non-SESE regions to constrain.

As it can be evinced from Figure 11, ϕ_{nSESE} works properly when there is at least one entry node of the non-SESE region $NR_{(\mathsf{ENset},\mathsf{EXset})}$ that either corresponds to the entry node of the minimal SESE region $R_{(en,ex)}$ enclosing $NR_{(\mathsf{ENset},\mathsf{EXset})}$ or it is directly connected to it (i.e., either en and en_1 coincide, or control flow edge (en, en_1) exists).

When there are process elements lying between the entry node of $R_{(en,ex)}$ and every entry node of the $NR_{(\mathsf{ENset},\mathsf{EXset})}$, duration pattern ϕ_{nSESE} must be slightly modified since the enactment of $NR_{(\mathsf{ENset},\mathsf{EXset})}$ is delayed with respect to the one of $R_{(en,ex)}$.

As an example, let us consider Figure 11(d). Starting from $\mathsf{G_{en}}$, both entry nodes $\mathsf{t_i}$ and $\mathsf{t_j}$ have predecessors, namely Other process elements 1 and Other process elements 2.

Since duration pattern ϕ_{nSESE} is anchored to $\mathsf{G_{en}}$, there must be a way of detecting when the first entry node of the constrained non-SESE region is enacted. To this end, a throwing signal event t_START is added to $R_{(\mathsf{G_{en}},\mathsf{G_{ex}})}$, immediately preceding each entry node of the non-SESE region. This event, delimits the boundary of the non-SESE region and triggers the corresponding signal event c_START placed before G1 and used to enable the whole duration pattern whose core is ϕ_{nSESE}.

We call $\phi_{nSESEPred}$ this variant of ϕ_{nSESE} able to handle predecessors with respect to the entry points of the considered non-SESE region. $\phi_{nSESEPred}$ can be connected to the minimal SESE enclosing the considered non-SESE through connecting kit $Ck_4 = (N_{k_4}, C_{k_4})$, where N_{k_4} contains one throwing signal event labeled t_START for each entry node of the non-SESE region having predecessors, and one event t_EXITED for each exit node of the non-SESE region, while $C_{k_4} = \varnothing$.

Overall, non-SESE regions having predecessors with respect to each entry node and multiple exit nodes can be handled by using a combination of signal events t_START and t_EXITED, connected to every entry (respectively exit) node of the region.

Figure 11(d) shows the duration-aware process model obtained by combining non-SESE region $NR_{(\mathsf{t_i},\mathsf{t_j})}$ with duration pattern $\phi_{nSESEPred}$ and connecting kit Ck_4 composed of four signal events, two labeled as t_START and two labeled as t_EXITED.

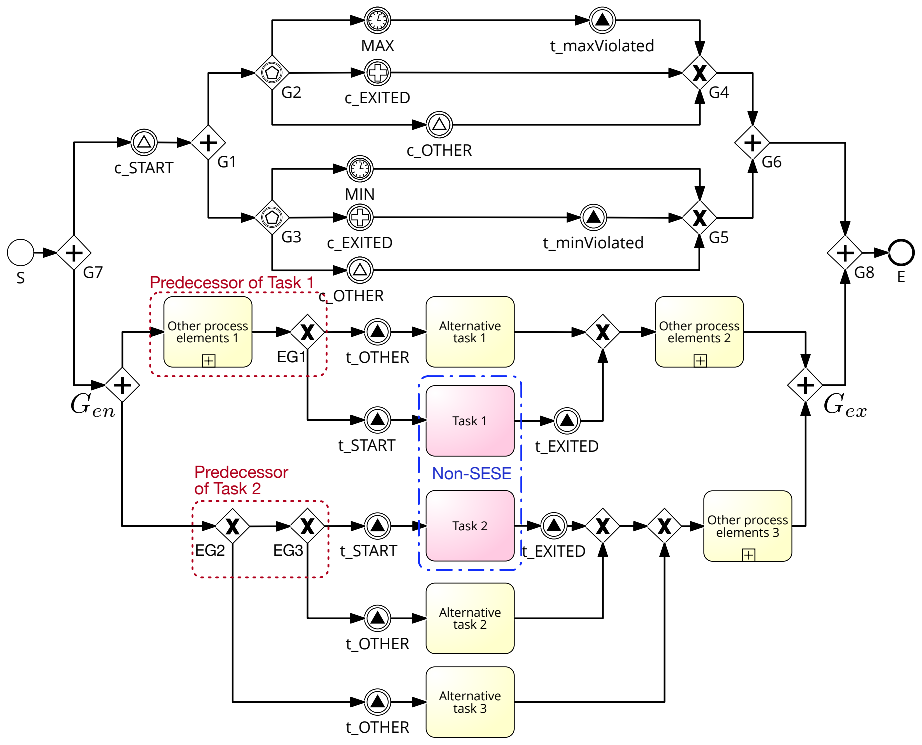


Figure 12: Duration-aware process model representing duration pattern $\phi_{nSESEGen}$ combined with a non-SESE region whose entry nodes have predecessors including exclusive gateways.

So far we have considered predecessors and successors as collapsed subprocesses, that is, as SESE regions. In this special scenario, all the process paths that start from the entry point of the minimal SESE $R_{(en,ex)}$ enclosing the non-SESE region of interest $NR_{(\mathsf{ENset},\mathsf{EXset})}$ reach also the entry points of the latter one.

However, when predecessors cannot be collapsed into SESE blocks, it is not always true that all process paths including en also include elements of ENset. In particular, when exclusive gateways in $R_{(en,ex)}$ precede the entry nodes of $NR_{(\mathsf{ENset},\mathsf{EXset})}$, process paths not leading to $NR_{(\mathsf{ENset},\mathsf{EXset})}$ may be chosen at run-time. Since in this case there is no need to continue with measuring duration, the duration pattern must be able to allow the process prosecuting its flow without taking any action.

As an example, consider the duration-aware process model shown in Figure 12. The non-SESE region $NR_{(\{\mathsf{Task1},\mathsf{Task2}\},\{\mathsf{Task1},\mathsf{Task2}\})}$ has multiple process elements preceding its entry and exit nodes, including exclusive gateways EG1, EG2, and EG3.

This setting is more challenging than those introduced above, as predecessors include alternative paths that do not belong to the non-SESE region but may be chosen before reaching the entry nodes of the region (paths outgoing from EG1, EG2, and EG3 not leading to Alternative Activity 1, Alternative Activity 2, and Alternative Activity 3). Thus, besides events t_START, denoting when the non-SESE regions is enacted, we need to use signal events t_OTHER denoting that a path not belonging to the non-SESE region has been chosen and, thus, the duration pattern is no more needed.

The following guidelines summarize how to place signal events to delimit the non-SESE region to constrain in the discussed scenario:

– a throwing signal event t_START must be placed before every entry node of the non-SESE region;
– a throwing signal event t_OTHER must be placed on every edge outgoing from an exclusive gateway located within the smallest enclosing SESE region and *alternative* to the non-SESE region to constrain;
– a throwing signal event t_EXITED must be placed after every exit node of the non-SESE region.

The placing of events t_OTHER is probably the trickiest one, as the concept of "alternative path" is transitive with respect to all nested exclusive gateways. That is, *all* paths that are alternative to the non-SESE region must be marked with a t_OTHER event.

As an example, consider again the process of Figure 12. The minimal SESE region $R_{(\mathsf{G_{en}},\mathsf{G_{ex}})}$ contains exclusive gateways EG1, EG2, and EG3 preceding the entry nodes of $NR_{(\{\mathsf{Task1},\mathsf{Task2}\},\{\mathsf{Task1},\mathsf{Task2}\})}$. Since both gateways EG2 and EG3 have outgoing paths that are alternative to Task 2, a t_OTHER event must be placed on each of these paths. Starting from duration pattern $\phi_{nSESEPred}$, two catching signal events c_OTHER, corresponding to all events t_OTHER are placed in the context of event-based gateways g_2 and g_3 to let tokens flowing ineffectively through the duration pattern every time an alternative path is chosen.

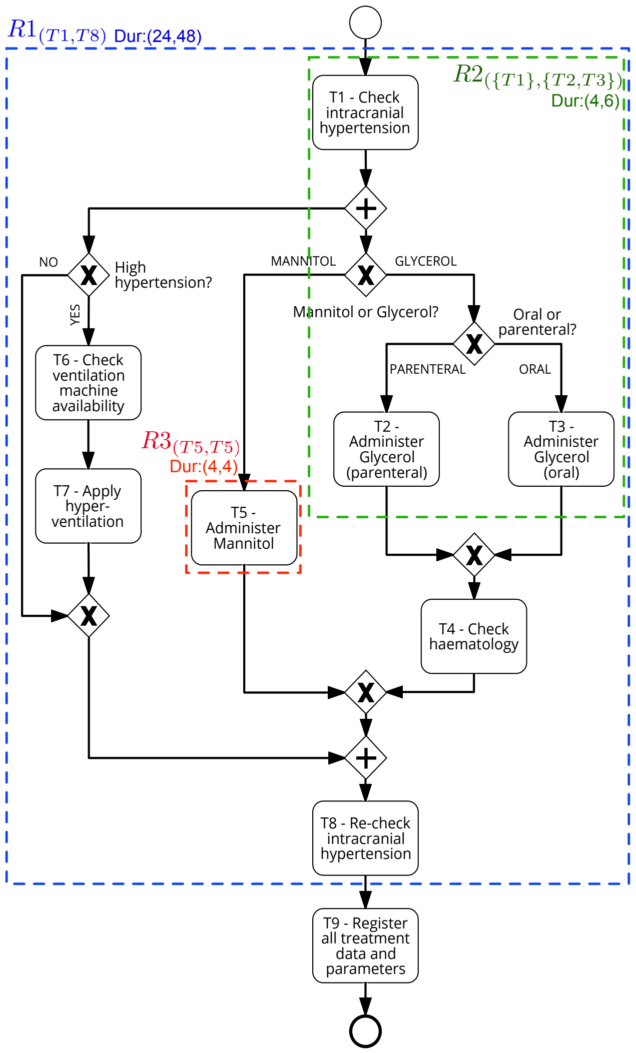
We call $\phi_{nSESEGen}$ this latest variant of duration pattern $\phi_{nSESEPred}$ as it is the more general duration pattern for handling duration of SESE regions having an arbitrary structure. Parallel gateways belonging to predecessors do not generate such problems, as the all paths outgoing from them are taken.

We intentionally leave out non-SESE regions that include part of looping structures, as their management is requires a specialized approach [31].

5.3. Composing duration patterns for process regions

Thanks to the composition properties of SESE regions [32], the proposed modular design approach allows one to (i) easily nest temporally constrained SESE regions within each other and (ii) specify the duration of any arbitrarily selected non-SESE region in the process. Indeed, in the worst case, the whole process would be the minimal enclosing SESE region for a certain multi entry or exit block.

In this paper we do not deal with duration constraints applied to process loops and other repetition structures, as their complexity would require a ded-



$R1_{(T1,T8)}$ Dur:(24,48)
$R2_{(\{T1\},\{T2,T3\})}$ Dur:(4,6)
$R3_{(T5,T5)}$ Dur:(4,4)
T1 - Check intracranial hypertension
NO
High hypertension?
YES
MANNITOL
GLYCEROL
Mannitol or Glycerol?
Oral or parenteral?
PARENTERAL
ORAL
T6 - Check ventilation machine availability
T7 - Apply hyper-ventilation
T5 - Administer Mannitol
T2 - Administer Glycerol (parenteral)
T3 - Administer Glycerol (oral)
T4 - Check haematology
T8 - Re-check intracranial hypertension
T9 - Register all treatment data and parameters

Figure 13: Simple process for the management of intercranial hypertension. SESE region R1 must last 24 to 48 hours; non-SESE region R2 must last 4 to 6 hours; SESE region R3 (task T5) must last exactly 4 hours.

icated approach. Therefore, the proposed duration pattern for addressing non-SESE regions does not apply to non-SESE regions containing parts of a loop beside other process elements. Instead, ϕ_{simple} can be used to specify the duration of complete loops forming SESE regions.

As a practical example, consider the process shown in Figure 13 describing a few basic steps for the management of intracranial hypertension, which include administration of glycerol, osmotherapy with mannitol, and hyperven-

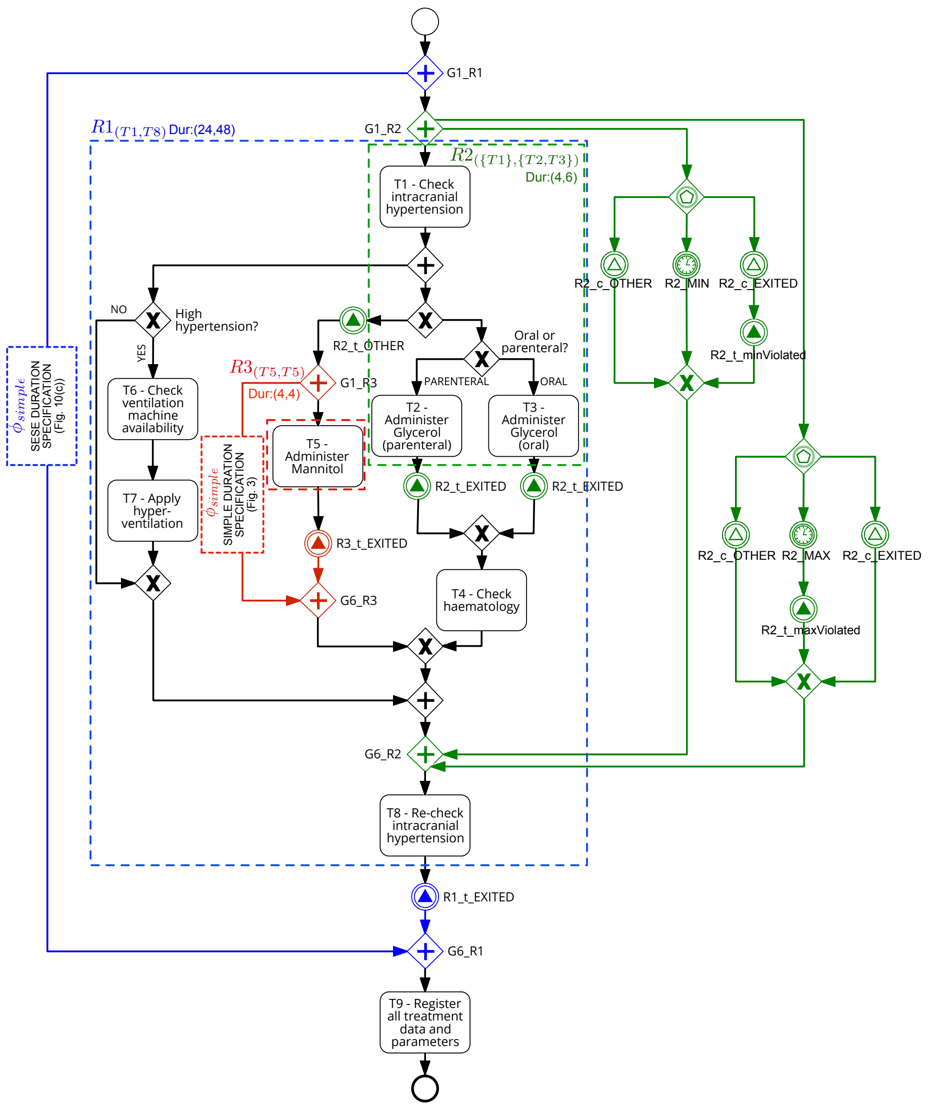

G1_R1
$R1_{(T1,T8)}$ Dur:(24,48)
G1_R2
$R2_{(\{T1\},\{T2,T3\})}$
Dur:(4,6)
T1 - Check intracranial hypertension
R2_c_OTHER
R2_MIN
R2_c_EXITED
R2_t_minViolated
NO
High hypertension?
YES
R2_t_OTHER
Oral or parenteral?
PARENTERAL
ORAL
$R3_{(T5,T5)}$
Dur:(4,4)
G1_R3
ϕ_{simple} SESE DURATION SPECIFICATION (Fig. 10(c))
T6 - Check ventilation machine availability
T2 - Administer Glycerol (parenteral)
T3 - Administer Glycerol (oral)
T5 - Administer Mannitol
ϕ_{simple} SIMPLE DURATION SPECIFICATION (Fig. 3)
T7 - Apply hyper-ventilation
R2_t_EXITED
R2_t_EXITED
R3_t_EXITED
R2_c_OTHER
R2_MAX
R2_c_EXITED
G6_R3
T4 - Check haematology
R2_t_maxViolated
G6_R2
T8 - Re-check intracranial hypertension
R1_t_EXITED
G6_R1
T9 - Register all treatment data and parameters

Figure 14: Example of attachment of duration patterns to process regions R1, R2, and R3.

tilation for treating severe patients [33]. Intracranial hypertension is a common neurologic complication in critically ill patients, which is usually managed with sedation, drainage of cerebrospinal fluid, and osmotherapy with mannitol. Mannitol should be administered for patients with elevated intracranial hypertension. Glycerol can also be prescribed and, during administration, haematology should be systematically checked since it may induce haemolysis.

Let us assume the following duration constraints. R1 is the SESE region starting with task T1 and ending with task T8 and must last 24 to 48 hours. R2 is the non-SESE region starting with T1 and ending with T2 or T3, depending on which administration route is chosen, and must last 4 to 6 hours. Finally, R3 is represented by task T5 which should last exactly 4 hours.

In order to specify the duration of R1, R2, and R3 we associate the most appropriate duration pattern to each region, by making sure that these are anchored correctly to the main process and by specializing signal events, so that the management of one process region does not affect the others. This approach facilitates the composition of process blocks, without compromising the generality of our solution. Indeed, the process model maintains the structural relationships between different regions and, of course, an expert designer can also decide to reduce the number of used signal events by combining them when regions share common starting/ending points.

Figure 14 shows where the duration patterns, and the signal events belonging to the related connecting kit can be positioned on the process of Figure 13 (the constructs for constraining R1 and R3 are only sketched in Figure 14, for clarity reasons). The duration patterns added for duration specification are not visible to final users as they may be collapsed in duration-aware subprocesses.

6. Specifying Deferred Activity Duration Constraints

In this section, we introduce the modeling of activity duration considering the case of having multiple duration ranges associated to one activity and assuming that the choice of which duration range applies is taken at run-time, that is, after activity initiation.

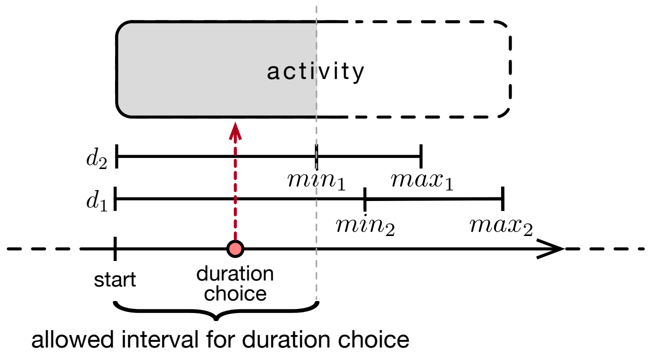


Figure 15: Example showing two different duration ranges d_1 and d_2 associated with one activity. The one that applies is chosen after activity initiation but at a moment in time preceding the smallest between min_1 and min_2.

In particular, we deal with a process activity that can be subjected to two or more alternative duration constraints $d_1, d_2, \ldots, d_n$. The choice of which among the constraints applies to the activity is made after its initiation, but prior to the smallest minimum duration bound among those of $d_1, d_2, \ldots, d_n$ and cannot change afterwards. Figure 15 shows the relationship between activity initiation, duration constraints, and duration choice. We refer to this kind of constraint as *deferred duration*, meaning that the choice of which duration range applies is deferred with respect to activity initiation.

As a motivating example, consider the following clinical setting. A patient is affected by an infection caused by *Staphylococcus Aureus*, a virulent and life-threatening bacterium that requires antibiotic treatment. The latter is administered taking into consideration the patient's allergies and bacteria resistance patterns. Typically, the therapy can last either 14 days or 4-6 weeks, depending on the extent of the infection. In particular, a 14-days therapy is administered only if fever disappears within 72 hours after treatment initiation and blood cultures are negative. However, the results of blood analyses are obtained 3 or 4 days after the beginning of the therapy. This means, that the choice of the required duration of the therapy is taken after empirical antibiotic administration has started (i.e., 3-4 days after therapy initiation), but earlier than the lowest minimum duration constraint (i.e., 14 days).

This meaningful example motivates the introduction and modeling of this novel kind of duration. Below, we formalize the structure of duration pattern $\phi_{deferred}$ considering two different duration ranges $[min_1, max_1]$ and $[min_2, max_2]$ associated to the activity to constrain.
Formally, $\phi_{deferred} = (N, C, \alpha, \epsilon_{tr}, \epsilon_{ty}, \beta, \delta, \gamma_r, \gamma_{ty}, \mathcal{L})$ has the following structure:

- $N = \{A \cup G \cup E\}$ is the set of flow nodes partitioned in: $A = \varnothing$; $G = \{g_1, g_{2min}, g_{2max}, g_{3min}, g_{3max}, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}, g_{11}, g_{12}, g_{13}, g_{14}\}$; $E = \{E_{start} \cup E_{int} \cup E_{end}\}$ where $E_{start} = \{s\}$, $E_{int} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_1^k, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}, e_{19}, e_{20}, e_{21}, e_{22}, e_{23}, e_{24}\}$, and $E_{end} = \{e\}$;
- $C = \{(s, g_1), (g_1, g_{2min}), (g_1, g_{2max}), (g_1, g_{3min}), (g_1, g_{3max}), (g_{3max}, e_1), (g_{3max}, e_2), (g_{3max}, e_3), (g_{3min}, e_4), (g_{3min}, e_5), (g_{3min}, e_6), (g_{2max}, e_7), (g_{2max}, e_8), (g_{2max}, e_9), (g_{2min}, e_{10}), (g_{2min}, e_{11}), (e_1, g_6), (g_6, e_{12}), (g_6, e_{15}), (e_{12}, e_{13}), (e_{13}, e_{14}), (e_{14}, g_9), (e_{15}, g_9), (g_9, g_{13}), (e_2, g_{13}), (e_3, g_{13}), (e_4, g_{12}), (e_5, e_{16}), (e_{16}, g_{12}), (e_6, g_{12}), (e_7, g_5), (g_5, e_{17}), (g_5, e_{20}), (e_{17}, e_{18}), (e_{18}, e_{19}), (e_{19}, g_8), (g_8, g_{11}), (g_{20}, g_8), (e_8, g_{11}), (e_9, g_{11}), (e_{10}, g_4), (g_4, e_{21}), (g_4, e_{23}), (e_{21}, e_{22}), (e_{22}, g_7), (e_{23}, g_7), (g_7, g_{10}), (e_{11}, g_{10}), (g_{13}, g_{14}), (g_{12}, g_{14}), (g_{11}, g_{14}), (g_{10}, g_{14}), (g_{14}, e)\}$;
- $\alpha = \varnothing$;
- $\epsilon_{tr}(\{s, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{15}, e_{17}, e_{20}, e_{21}, e_{23}\}) = catching$, $\epsilon_{tr}(\{e_{13}, e_{14}, e_{16}, e_{18}, e_{19}, e_{22}, e_{24}, e\}) = throwing$;
- $\epsilon_{ty}(\{s, e\}) = none$, $\epsilon_{ty}(\{e_1, e_4, e_7, e_{10}\}) = timer$, $\epsilon_{ty}(\{e_2, e_3, e_5, e_6, e_8, e_9, e_{11}, e_{13}, e_{14}, e_{16}, e_{18}, e_{19}, e_{22}, e_{24}\}) = signal$, $\epsilon_{ty}(\{e_{12}, e_{15}, e_{17}, e_{20}, e_{21}, e_{23}\}) = conditional$;
- $\beta = \varnothing$;
- $\delta = \varnothing$;
- $\gamma_r(\{g_1, g_{2min}, g_{2max}, g_{3min}, g_{3max}, g_4, g_5, g_6\} = split$, $\gamma_r(\{g_7, g_8, g_9, g_{10}, g_{11}, g_{12}, g_{13}, g_{14}\} = merge$;
- $\gamma_{ty}(\{g_1, g_{14}\} = parallel$, $\gamma_{ty}(\{g_{2min}, g_{2max}, g_{3min}, g_{3max}, g_4, g_5, g_6\} = event{-}based$ $\gamma_{ty}(\{g_7, g_8, g_9, g_{10}, g_{11}, g_{12}, g_{13}\} = exclusive$;
- $\mathcal{L}(s) =$ S, $\mathcal{L}(g_1) =$ G1, $\mathcal{L}(g_{2min}) =$ G2_MIN, $\mathcal{L}(g_{2max}) =$ G2_MAX, $\mathcal{L}(g_{3min})$

$=$ G3_MIN, $\mathcal{L}(g_{3max})$ = G3_MAX, $\mathcal{L}(g_4)$ = G4, $\mathcal{L}(g_5)$ = G5, $\mathcal{L}(g_6)$ = G6, $\mathcal{L}(g_7)$ = G7, $\mathcal{L}(g_8)$ = G8, $\mathcal{L}(g_9)$ = G9, $\mathcal{L}(g_{10})$ = G10, $\mathcal{L}(g_{11})$ = G11, $\mathcal{L}(g_{12})$ = G12, $\mathcal{L}(g_{13})$ = G13, $\mathcal{L}(g_{14})$ = G14, $\mathcal{L}(e_1)$ = *null*, $\mathcal{L}(e_2)$ = c_EXITED, $\mathcal{L}(e_3)$ = c_cancelMax, $\mathcal{L}(e_4)$ = *null*, $\mathcal{L}(e_5)$ = c_EXITED, $\mathcal{L}(e_6)$ = c_cancelMin, $\mathcal{L}(e_7)$ = *null*, $\mathcal{L}(e_8)$ = c_EXITED, $\mathcal{L}(e_9)$ = c_cancelMax, $\mathcal{L}(e_{10})$ = *null*, $\mathcal{L}(e_{11})$ = c_EXITED, $\mathcal{L}(e_{12})$ = *null*, $\mathcal{L}(e_{13})$ = t_cancelMax, $\mathcal{L}(e_{14})$ = t_maxViolated, $\mathcal{L}(e_{15})$ = *null*, $\mathcal{L}(e_{16})$ = t_minViolated, $\mathcal{L}(e_{17})$ = *null*, $\mathcal{L}(e_{18})$ = t_cancelMax, $\mathcal{L}(e_{19})$ = t_maxViolated, $\mathcal{L}(e_{20})$ = *null*, $\mathcal{L}(e_{21})$ = *null*, $\mathcal{L}(e_{22})$ = t_cancelMin, $\mathcal{L}(e_{23})$ = *null*, $\mathcal{L}(e_{24})$ = t_minViolated, $\mathcal{L}(e)$ = E.

It is worth noticing that the two duration ranges captured by $\phi_{deferred}$ are not required to be disjoint, as only one of them can be chosen at a time.

We reuse connecting kit $Ck_1 = (N_{k_1}, C_{k_1})$ where $N_{k_1} = \{e_1^k\}$ and $C = \varnothing$. Event e_1^k triggers all four events e_2, e_5, e_8, and e_{11} of $\phi_{deferred}$ whenever they are actively listening (indeed $\mathcal{L}(e_2) = \mathcal{L}(e_5) = \mathcal{L}(e_8) = \mathcal{L}(e_{11})$).

The formal construct for specifying that the deferred duration of an activity named ActivityName has to be either between MIN1 and MAX1 or MIN2 and MAX2 time units, depending on whether condition C1 or C2 holds is defined as

$$\texttt{DeferredDur}(\text{ActivityName}, \text{MIN1}, \text{MAX1}, \text{MIN2}, \text{MAX2}, \text{C1}, \text{C2})$$

It generates a (sub)process model by using $\phi_{deferred}$ and Ck_1 as follows:

DeferredDur(ActivityName, MIN1, MAX1, MIN2, MAX2, C1, C2) = $(N \cup N_{k_1} \cup \{a\}, C \cup \{(g_1, a), (a, e_1^k), (e_1^k, g_{14})\}, \alpha \cup \{\{a \mapsto task\}$ or $\{a \mapsto subprocess\}\}, \epsilon_{tr} \cup \{e_1^k \mapsto throwing\}, \epsilon_{ty} \cup \{e_1^k \mapsto signal\}, \beta, \delta, \gamma_r, \gamma_{ty}, \mathcal{L}^*)$.

The new labeling function $\mathcal{L}^*$ extends $\mathcal{L}$ for e_1, e_4, e_7, e_{10}, e_{12}, e_{15}, e_{17}, e_{20}, e_{21}, e_{23}, e_1^k, and a, where $\mathcal{L}^*(e_1)$ = MAX2, $\mathcal{L}^*(e_4)$ = MIN2, $\mathcal{L}^*(e_7)$ = MAX1, $\mathcal{L}^*(e_{10})$ = MIN1, $\mathcal{L}^*(\{e_{15}, e_{17}, e_{21}\})$ = C1, $\mathcal{L}^*(\{e_{12}, e_{20}, e_{23}\})$ = C2, $\mathcal{L}^*(e_1^k)$ = t_EXITED, and $\mathcal{L}^*(a)$ = ActivityName.

In the duration-aware process model of Figure 16 the considered activity is represented as a task, i.e., $\alpha(a) = task$ and $\mathcal{L}^*(a)$ = Task, without loss of generality.

For modeling Task deferred duration, we should take four values into consideration, which represent the minimum and maximum extremes of the two allowed duration values, i.e., $d_1 \in$ [MIN1, MAX1] and $d_2 \in$ [MIN2, MAX2], but the same solution can be generalized to deal with more than two duration ranges.

The intuition behind the behavior of this process relies on the assumption that among the four duration extremes MIN1, MAX1, MIN2 and MAX2 there is always a smallest one, regardless of how the two duration ranges are related to each other (If MIN1 = MIN2, then we can arbitrarily select one of them). This lowest minimum duration value is needed for ordering duration ranges based on the value of their minimum duration since it is crucial to know the latest time-instant at which Task duration must be chosen.

Without loss of generality, let us suppose that d_1 is the time length having the lowest minimum duration (MIN1 in Figure 16).

The behavior of the process can be explained as follows. After start event S is triggered, parallel gateway G1 splits the flow into five branches, one directed

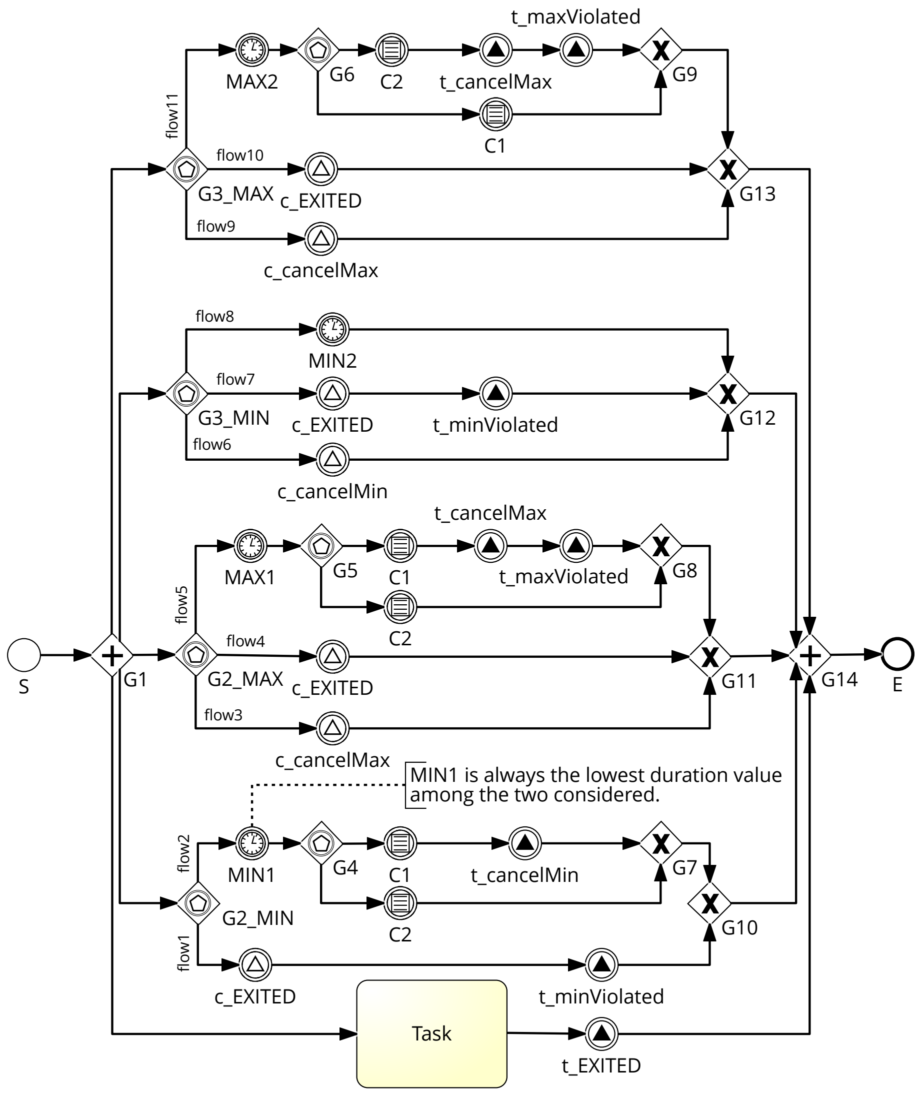


Figure 16: Duration-aware process model for specifying that the deferred duration of Task may either be $d_1 \in [\mathsf{MIN1}, \mathsf{MAX1}]$ or $d_2 \in [\mathsf{MIN2}, \mathsf{MAX2}]$. The choice (e.g., C1 or C2) of which between d_1 and d_2 applies is deferred after task initiation.

towards Task and the others ones leading to event-based gateways G2_MIN, G2_MAX, G3_MIN, and G3_MAX.

From this point on, the process behavior depends on the moment of Task completion, on which duration range is chosen, and on the kind (if any) of violated duration (i.e., minimum or maximum).

Duration choice is represented by conditional events which are triggered by the environment whenever a certain condition is true [13]. In general, anything can be a condition and conditions are independent of processes. Conditional events exist only of type catching and are triggered when a data-based condition evaluates to true [18]. In this paper, we assume that when a conditional event is enabled by a token, the process checks whether the associated condition is true and, if so, the event is triggered. Otherwise, the token will remain on the event (i.e., the event will remain enabled) until the associated condition becomes true. In particular, if d_1 is the chosen duration, then the condition associated to conditional event C1 is true, otherwise the condition associated to conditional event C2 holds and d_2 is chosen.

In the described scenario, minimum duration is violated when:

(a) Task completes before its desired duration is chosen;

(b) Task ends before the lowest minimum value set for duration, that is MIN1, when the chosen duration is d_1;

(c) Task completes before MIN2 when the chosen duration is d_2.

(a) If Task completes earlier than knowing which is the chosen duration range, signal event t_EXITED is thrown to be caught by all the four corresponding events c_EXITED located on flow1, flow4, flow7, and flow10. Then, the violation of minimum duration constraint is signaled by both events t_minViolated on flow1 and flow7 before the whole process can complete. Since duration range has not yet been chosen, all minimum durations are violated.

(b) If Task completes earlier than MIN1, being MIN1 the smallest admissible duration value, a violation has to be signaled regardless of which is the chosen duration range. In this case, the process behaves as discussed in (a), as signal event t_EXITED is caught by all the corresponding listening events.

(c) If Task completes earlier than MIN2, a violation must be signaled only if the chosen duration range is d_2. Event-based gateway G4, enacted right after event MIN1, differentiates the process behavior with respect to which duration range has been chosen. If Task duration is d_1, no minimum violation is signaled: signal event t_cancelMin is thrown to trigger the corresponding event c_cancelMin on flow6, thus discarding timer event MIN2.

If Task duration is d_2, then a minimum duration violation is observed. In particular, on flow2 timer event MIN1 has already fired as MIN1 $\leq$ MIN2. In the configuration of event-based gateway G4, conditional event C2 is triggered to let the process flow proceeding until exclusive gateway G7 without further signaling. The detection of the activity early completion is handled by signal events c_EXITED and t_minViolated located within flow7. Similarly, it is worth noticing that in case $d_1 < d_2$, also event C2 in the scope of G5 is triggered, as there is no need to check duration MAX1.

For all the discussed cases (a)–(c), the process flows branching from G2_MAX

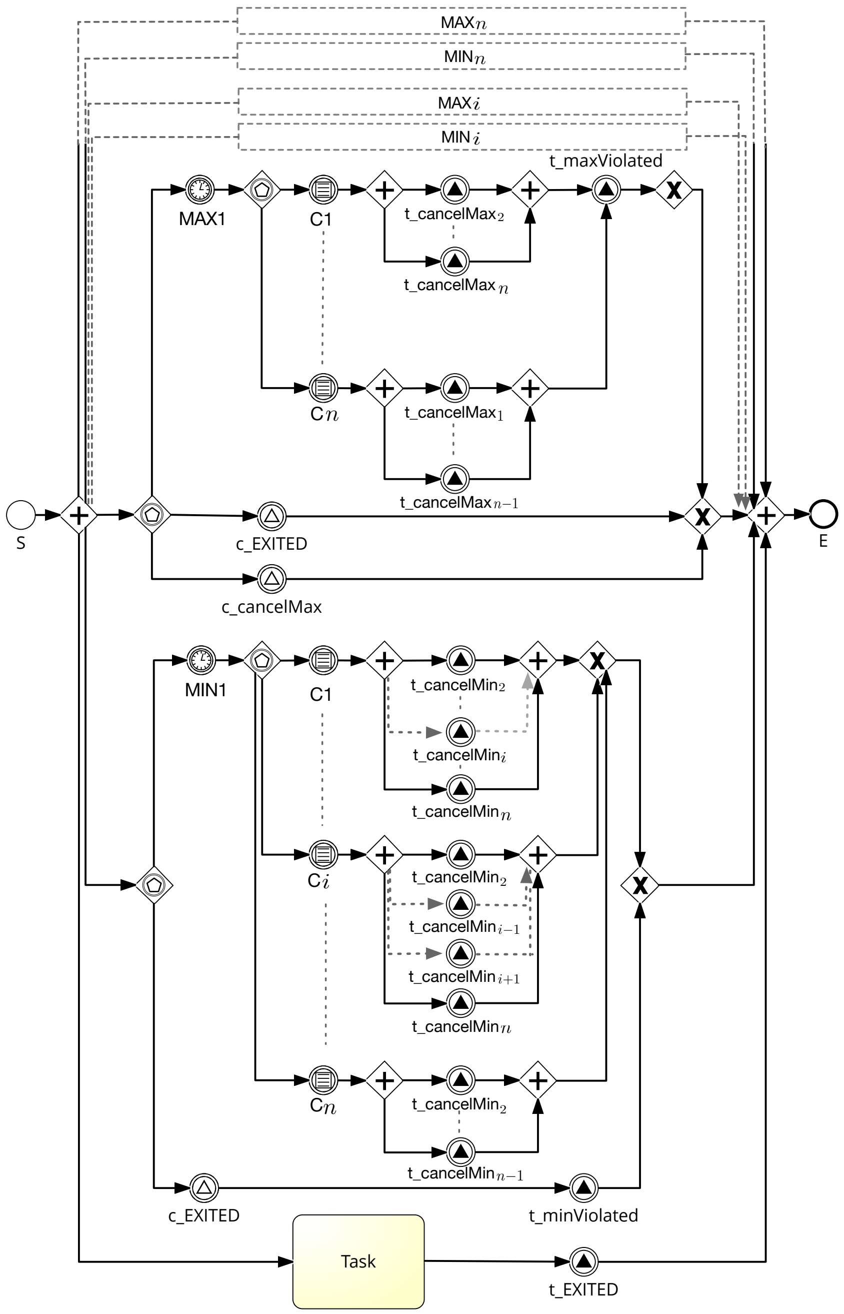


Figure 17: Complete process combining control structures for capturing deferred duration which is chosen among an arbitrary number of duration ranges.

and G3_MAX have the following behavior. Regardless of which is the maximum duration considered, if the corresponding timer event has not yet been triggered, then either signal events c_EXITED on flow4 and flow10 or signal events c_cancelMax on flow3 and flow9 will be triggered letting the flow proceed without any further signaling.

When considering violations of maximum duration the process behaves as follows. Despite durations d_1 and d_2 are ordered according to the lowest value for minimum duration, timer event MAX1 may be assigned a duration value smaller, equal, or greater than the one of timer event MAX2.

If the chosen duration is d_1 the condition associated to conditional event C1 is true. When maximum duration is violated, signal event t_cancelMax following event-based gateway G5 and event C1 is thrown to "cancel" event handlers for d_2. In particular, if MAX1 $<$ MAX2, the triggered signal is caught by signal event c_cancelMax on flow9. Then, event t_maxViolated on flow5 indicates that the maximum duration has been violated. If MAX1 $>$ MAX2, then signal c_cancelMax is never caught as flow11 has been chosen.

Besides the discussed cases, we consider the possibility that either (i) MIN1 = MIN2, or (ii) MAX1 = MAX2. We prove that the model behaves soundly for the mentioned cases as follows.

(i) If MIN1 = MIN2, the lowest minimum can be chosen arbitrarily. A minimum duration violation will involve both d_1 and d_2, as for the previously discussed case (b).

(ii) If MAX1 = MAX2, signal event t_cancelMax, thrown after the timer event representing the chosen maximum duration limit, is never caught. In detail, if we choose d_1 as the preferred Task duration, even if timer event MAX2 is triggered concurrently to MAX1, conditional events C1 and C2 in the scope of event-based gateway G4 ensure that signal t_maxViolated is triggered only once. Specular behavior is expected if d_2 is chosen.

When dealing with more than two duration ranges, signal events labeled t_cancelMin, c_cancelMin, t_cancelMax, and c_cancelMax in Figure 16 must be specialized into t_cancelMin$_i$, c_cancelMin$_i$, t_cancelMax$_i$, and c_cancelMax$_i$ in order to let the process flow through the branches designed to represent duration ranges D$_i$ that are not chosen by the activity. Besides, one conditional event C$_i$ must be added for each possibly chosen duration d_i.

Figure 17 sketches how duration pattern $\phi_{deferred}$ can be extended with specialized conditional and signal events to handle more than two duration ranges.

7. Specifying Shifted Duration Constraints

In this section, we introduce patterns for specifying shifted duration constraints that extend and complete the preliminary proposal described in [19]. Shifting the activity duration means that duration is measured only after a certain condition is met, i.e., the duration of the activity is evaluated starting from a particular moment in time that is *shifted* forward in time with respect to activity initiation.

As an example, let us consider the treatment of uncomplicated pneumococcal pneumonia, a common lung infection leading to hospitalization. Antibiotic therapy must be initiated immediately after the onset of the infection. Then, depending on the extent of the infection and the patient clinical response, antibiotic administration is continued for 5–7 days after defervescence. The overall duration of therapy should not exceed 10 days. If the patient does not defervesce within 3–5 days, antibiotic susceptibility must be revised.

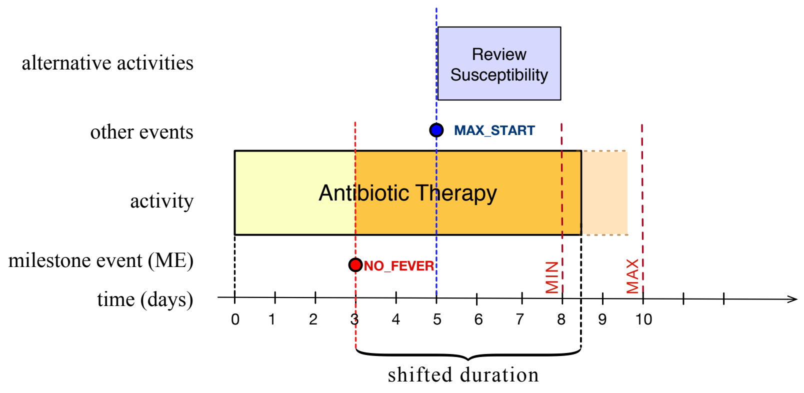


Figure 18: Shifted duration exemplified with respect to activity Antibiotic Therapy to treat pneumococcal pneumonia.

The concept of shifted duration for the introduced example is shown in Figure 18. Antibiotic Therapy begins, but its duration $d \in [5, 7]$ days is measured starting from *milestone event (ME)* NO_FEVER. If the latter does not occur within 5 days (the maximum time allowed for defervescence), event MAX_START triggers *alternative activity* Review Susceptibility.

To represent shifted duration of an activity, we designed duration pattern $\phi_{shifted} = (N, C, \alpha,\ \epsilon_{tr},\ \epsilon_{ty},\ \beta,\ \delta,\ \gamma_r, \gamma_{ty},\ \mathcal{L})$ a process fragment which is formalized as follows:

- $N = \{A \cup G \cup E\}$ is the set of flow nodes, where: $A = \{a_1\}$; $G = \{g_2,\ g_3,\ g_4,\ g_5,\ g_6,\ g_7, g_8, g_9\}$; $E = \{E_{start} \cup E_{int} \cup E_{end}\}$ where $E_{start} = \{s\}$, $E_{int} = \{e_1,\ e_2,\ e_3,\ e_4,\ e_5,\ e_6,\ e_7,\ e_8,\ e_9,\ e_{10}\}$, and $E_{end} = \{e\}$;
- $C = \{(s, g_1), (g_1, g_2)\ (g_2, e_1), (g_2, e_2), (g_2, e_3), (e_3, e_4), (e_4, g_9), (e_1, a_1), (a_1, g_9), (e_2, g_3), (g_3, g_4), (g_3, g_5), (g_4, e_5), (g_4, e_6), (e_6, g_6), (g_5, e_7), (g_5, e_8), (e_5, e_9), (e_9, g_6), (g_6, g_8), (e_8, e_{10}), (e_7, g_7), (e_{10}, g_7), (g_7, g_8), (g_8, g_9), (g_9, g_{10}), (g_{10}, e)\}$ is the set of control flow edges;
- $\alpha(a_1) = \textit{subprocess}$;
- $\epsilon_{tr}(\{s, e_1, e_2, e_3, e_5, e_6, e_7, e_8\}) = \textit{catching}$, $\epsilon_{tr}(\{e_4, e_9, e_{10}, e\}) = \textit{throwing}$;
- $\epsilon_{ty}(\{s, e\}) = \textit{none}$, $\epsilon_{ty}(\{e_1, e_5, e_7\}) = \textit{timer}$, $\epsilon_{ty}(\{e_3, e_4, e_6, e_8, e_9, e_{10}\}) = \textit{signal}$, $\epsilon_{ty}(\{e_2\}) = \textit{conditional}$;
- $\beta = \varnothing$;
- $\delta = \varnothing$;
- $\gamma_r(\{g_1, g_2, g_3, g_4, g_5\}) = \textit{split}$, $\gamma_r(\{g_6, g_7, g_8, g_9, g_{10}\}) = \textit{merge}$;

– $\gamma_{ty}(\{g_1, g_3, g_8, g_{10}\}) = parallel$, $\gamma_{ty}(\{g_6, g_7, g_9\}) = exclusive$, and $\gamma_{ty}(\{g_2, g_4, g_5\}) = event\text{-}\ based$;
– $\mathcal{L}(a_1) =$ *null*, $\mathcal{L}(s) =$ S, $\mathcal{L}(e_1) =$ *null*, $\mathcal{L}(e_2) =$ *null*, $\mathcal{L}(e_3) =$ c_EXITED, $\mathcal{L}(e_4)$ = t_minViolated, $\mathcal{L}(e_5) =$ *null*, $\mathcal{L}(e_6) =$ c_EXITED, $\mathcal{L}(e_7) =$ *null*, $\mathcal{L}(e_8) =$ c_EXITED, $\mathcal{L}(e_9) =$ t_maxViolated, $\mathcal{L}(e_{10}) =$ t_minViolated, $\mathcal{L}(e) =$ E, $\mathcal{L}(g_1) =$ G1, $\mathcal{L}(g_2) =$ G2, $\mathcal{L}(g_3) =$ G3, $\mathcal{L}(g_4) =$ G4, $\mathcal{L}(g_5) =$ G5, $\mathcal{L}(g_6) =$ G6, $\mathcal{L}(g_7) =$ G7, $\mathcal{L}(g_8) =$ G8, $\mathcal{L}(g_9) =$ G9, $\mathcal{L}(g_{10}) =$ G10.

We reuse connecting kit $Ck_1 = (N_{k_1}, C_{k_1})$, where $N_{k_1} = \{e_1^k\}$ and $C_{k_1} = \varnothing$. Event e_1^k triggers all three events e_3, e_6, and e_8 of $\phi_{shifted}$ whenever they are actively listening (indeed $\mathcal{L}(e_3) = \mathcal{L}(e_6) = \mathcal{L}(e_8)$).

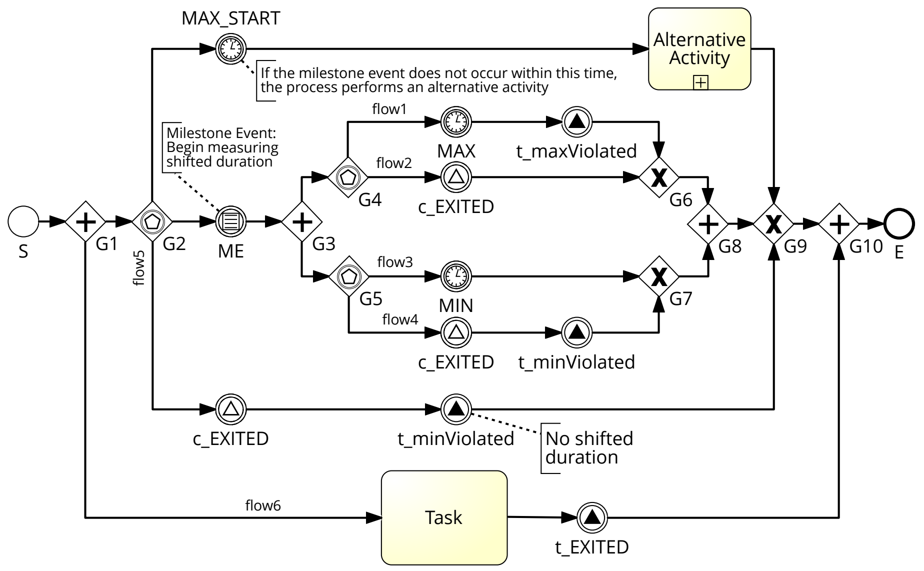


Figure 19: Duration-aware process model for specifying the shifted duration of a Task.

Below we consider specifying the shifted duration of an activity named ActivityName has to be between MIN and MAX time units, which are counted starting from the occurrence of milestone event ME. If ME does not occur within a certain time limit MAX_START, then alternative activity AltActivity is conducted. The formal construct is defined as:

ShiftedDur(ActivityName, MIN, MAX, ME, MAX_START, AltActivity)

It generates a (sub)process model by using $\phi_{shifted}$ and Ck_1 as follows:

ShiftedDur(ActivityName, MIN, MAX, ME, MAX_START, AltActivity) $= (N \cup N_{k_1} \cup \{a\}, C \cup \{(g_1, a), (a, e_1^k), (e_1^k, g_{10})\}, \alpha \cup \{\{a \mapsto task\}$ or $\{a \mapsto subprocess\}\}, \epsilon_{tr} \cup \{e_1^k \mapsto throwing\}, \epsilon_{ty} \cup \{e_1^k \mapsto signal\}, \beta, \delta, \gamma_r, \gamma_{ty}, \mathcal{L}^*)$.

The new labeling function $\mathcal{L}^*$ extends $\mathcal{L}$ for a, a_1, e_1, e_2, e_5, e_7, and e_1^k, where $\mathcal{L}^*(a)$ = ActivityName, $\mathcal{L}^*(a_1)$ = AltActivity, $\mathcal{L}^*(e_1)$ = MAX_START, $\mathcal{L}^*(e_2)$ = ME, $\mathcal{L}^*(e_5)$ = MAX, $\mathcal{L}^*(e_7)$ = MIN, and $\mathcal{L}^*(e_1^k)$ = t_EXITED.

Figure 19 depicts the complete duration-aware process model for specifying the shifted duration of an activity, obtained through ShiftedDur(Task, MIN, MAX, ME, MAX_START, AlternativeActivity). Without loss of generality, the considered activity is represented as a task, i.e., $\alpha(a) = task$ and $\mathcal{L}^*(a)$ = Task.

It is worth noticing that both patterns ϕ_{simple} and $\phi_{shifted}$ are both attached to the activity to be constrained through connecting kit Ck_1.

The complete duration-aware process model, shown in Figure 19 behaves as follows. Once the process is started, the flow is split by G1 into two branches: flow6 is directed towards Task, which can begin its execution, while on the other branch event-based gateway G2 is enabled. This realizes a race condition between the occurrence of timer event MAX_START, which ensures that something alternative is done if the milestone event does not occur, milestone event ME, and signal event c_EXITED, which handles the case in which Task ends before any of the other two events has occurred. In this latter "borderline scenario", minimum violation occurs, as captured by the follow signal event t_minViolated, but the measurement of shifted duration has not yet started. As mentioned in Section 6, conditional event ME can fire when it is enabled by the incoming token and the associated condition is true.

The detection of shifted duration violations is realized by using a simple signal-based communication pattern. When Task completes its execution, signal event t_EXITED is broadcast to be caught by any of the corresponding c_EXITED events that are active at the moment of broadcasting. Based on when Task ends, any of the following mutually exclusive scenarios can occur.

Minimum shifted duration is violated if Task completes earlier than MIN. In this case, both signals c_EXITED are triggered, whereas the other process branches outgoing of event-based gateways G4 and G5 are withdrawn. Then, signal event t_minViolated, is used to capture the violation. Finally, once synchronization has occurred at G10, the process can conclude.

If Task ends anytime between MIN and MAX no violation occurs and signal event t_EXITED is caught only by the corresponding c_EXITED on flow2, as timer event MIN has already fired.

Maximum shifted duration is violated whenever Task execution lasts longer than MAX. In this case, right after MAX, signal t_maxViolated is broadcast to detect that maximum shifted duration has been violated. Trivially, as timer event MIN had also been triggered before MAX, the process can complete once synchronization has occurred at G10.

In [19], we also propose a pattern for specifying a shifted duration with reset. Indeed, ME captures the beginning of a certain condition that must hold for the whole period of shifted duration. When such condition is not more valid, we must capture the change and reset shifted duration.

For example, let us consider hospitalization and discharge from hospital. Discharge criteria require patients to have been afebrile for at least 24 hours to be safely dismissed [19]. This means that, whenever fever (re-)appears the patient must wait for temperature to lower within normal ranges and, from that moment, stay in hospital for 24 fever-free additional hours. To specify the shifted duration of activity "hospitalization", which should be of at least 24 hours after "fever disappears", our milestone event we should consider reset condition "fever comes back", which requires physicians to wait until fever disappears again before re-counting 24 hours.

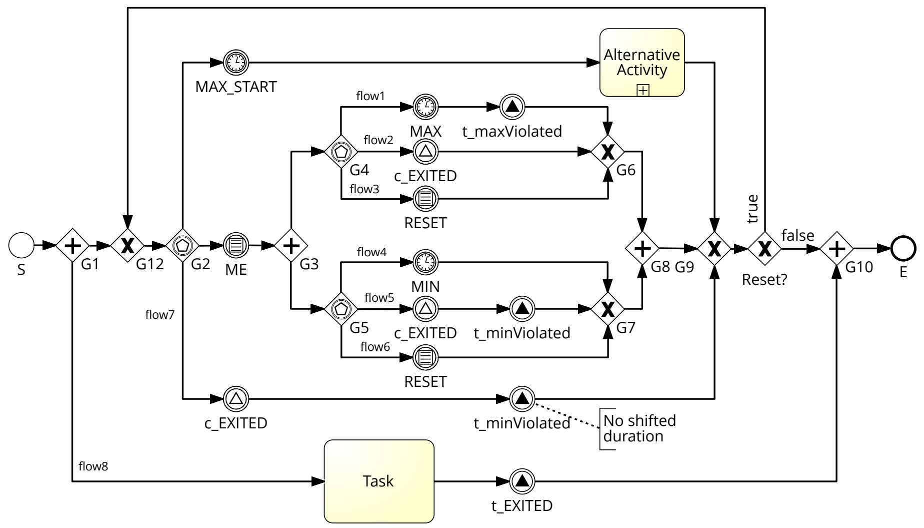


Figure 20: Process model for specifying the shifted duration of a task with reset.

Formally, the structure of duration pattern $\phi_{shiftRes} = (N, C, \alpha, \epsilon_{tr}, \epsilon_{ty}, \beta, \delta, \gamma_r, \gamma_{ty}, \mathcal{L})$ for specifying shifted duration with reset is as follows.

- $N = \{A \cup G \cup E\}$ is the set of flow nodes, where: $A = \{a_1\}$; $G = \{g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9, g_{10}, g_{11}, g_{12}\}$; $E = \{E_{start} \cup E_{int} \cup E_{end}\}$ where $E_{start} = \{s\}$, $E_{int} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$, and $E_{end} = \{e\}$;
- $C = \{(s, g_1), (g_1, g_{12}), (g_{12}, g_2), (g_2, e_1), (g_2, e_2), (g_2, e_3), (e_3, e_4), (e_4, g_9), (e_1, a_1), (a_1, g_9), (e_2, g_3), (g_3, g_4), (g_3, g_5), (g_4, e_5), (g_4, e_6), (g_4, e_7), (e_6, g_6), (e_7, g_6), (g_5, e_8), (g_5, e_9), (g_5, e_{10}), (e_5, e_{11}), (e_{11}, g_6), (g_6, g_8), (e_9, e_{12}), (e_8, g_7), (e_{12}, g_7), (e_{10}, g_7), (g_7, g_8), (g_8, g_9), (g_9, g_{11}), (g_{11}, g_{10}), (g_{11}, g_{12}), (g_{10}, e)\}$ is the set of control flow edges;
- $\alpha(a_1) = \textit{subprocess}$;
- $\epsilon_{tr}(\{s, e_1, e_2, e_3, e_5, e_6, e_7, e_8, e_9, e_{10}\}) = \textit{catching}$, $\epsilon_{tr}(\{e_4, e_{11}, e_{12}, e\}) = \textit{throwing}$;
- $\epsilon_{ty}(\{s, e\}) = \textit{none}$, $\epsilon_{ty}(\{e_1, e_5, e_8\}) = \textit{timer}$, $\epsilon_{ty}(\{e_3, e_4, e_6, e_9, e_{11}, e_{12}\}) = \textit{signal}$, $\epsilon_{ty}(\{e_2, e_7, e_{10}\}) = \textit{conditional}$;
- $\beta = \varnothing$;
- $\delta = \varnothing$;
- $\gamma_r(\{g_1, g_2, g_3, g_4, g_5, g_{11}\}) = \textit{split}$, $\gamma_r(\{g_6, g_7, g_8, g_9, g_{10}, g_{12}\}) = \textit{merge}$;

– $\gamma_{ty}(\{g_1, g_3, g_8, g_{10}\}) = \textit{parallel}$, $\gamma_{ty}(\{g_6, g_7, g_9, g_{11}, g_{12}\}) = \textit{exclusive}$, and $\gamma_{ty}(\{g_2, g_4, g_5\}) = \textit{event-- based}$;
– $\mathcal{L}(a_1) = \textit{null}$, $\mathcal{L}(s) = \mathsf{S}$, $\mathcal{L}(e_1) = \textit{null}$, $\mathcal{L}(e_2) = \textit{null}$, $\mathcal{L}(e_3) = \mathsf{c\_EXITED}$, $\mathcal{L}(e_4) = \mathsf{t\_minViolated}$, $\mathcal{L}(e_5) = \textit{null}$, $\mathcal{L}(e_6) = \mathsf{c\_EXITED}$, $\mathcal{L}(e_7) = \textit{null}$, $\mathcal{L}(e_8) = \textit{null}$, $\mathcal{L}(e_9) = \mathsf{c\_EXITED}$, $\mathcal{L}(e_{10}) = \textit{null}$, $\mathcal{L}(e_{11}) = \mathsf{t\_maxViolated}$, $\mathcal{L}(e_{12}) = \mathsf{t\_minViolated}$, $\mathcal{L}(e) = \mathsf{E}$, $\mathcal{L}(g_1) = \mathsf{G1}$, $\mathcal{L}(g_2) = \mathsf{G2}$, $\mathcal{L}(g_3) = \mathsf{G3}$, $\mathcal{L}(g_4) = \mathsf{G4}$, $\mathcal{L}(g_5) = \mathsf{G5}$, $\mathcal{L}(g_6) = \mathsf{G6}$, $\mathcal{L}(g_7) = \mathsf{G7}$, $\mathcal{L}(g_8) = \mathsf{G8}$, $\mathcal{L}(g_9) = \mathsf{G9}$, $\mathcal{L}(g_{10}) = \mathsf{G10}$, $\mathcal{L}(g_{11}) = \mathsf{Reset?}$, and $\mathcal{L}(g_{12}) = \mathsf{G12}$.

We reuse connecting kit $Ck_1 = (N_{k_1}, C_{k_1})$, where $N_{k_1} = \{e_1^k\}$ and $C_{k_1} = \varnothing$ Event e_1^k triggers all three events e_3, e_6, and e_9 of $\phi_{shiftRes}$ whenever they are actively listening (indeed $\mathcal{L}(e_3) = \mathcal{L}(e_6) = \mathcal{L}(e_9)$).

The formal construct to specify the shifted duration of an activity ActivityName with the possibility of resetting the duration count in case of a reset event R is defined as:

`ShiftedDurR`(ActivityName, MIN, MAX, ME, MAX_START, AltActivity, R)

It generates a (sub)process model by using $\phi_{shiftRes}$ and Ck_1 as follows:

`ShiftedDurR`(ActivityName, MIN, MAX, ME, MAX_START, AltActivity, R) = $(N \cup N_{k_1} \cup \{a\}, C \cup \{(g_1, a), (a, e_1^k), (e_1^k, g_{10})\}, \alpha \cup \{\{a \mapsto \textit{task}\}$ or $\{a \mapsto \textit{subprocess}\}\}, \epsilon_{tr} \cup \{e_1^k \mapsto \textit{throwing}\}, \epsilon_{ty} \cup \{e_1^k \mapsto \textit{signal}\}, \beta, \delta, \gamma_r, \gamma_{ty}, \mathcal{L}^*)$.

The new labeling function $\mathcal{L}^*$ extends $\mathcal{L}$ for a, a_1, e_1, e_2, e_5, e_7, e_8, e_{10}, and e_1^k where $\mathcal{L}^*(a_1) = \mathsf{AltActivity}$, $\mathcal{L}^*(e_1) = \mathsf{MAX\_START}$, $\mathcal{L}^*(e_2) = \mathsf{ME}$, $\mathcal{L}^*(e_5) = \mathsf{MAX}$, $\mathcal{L}^*(e_7) = \mathsf{R}$, $\mathcal{L}^*(e_8) = \mathsf{MIN}$, $\mathcal{L}^*(e_{10}) = \mathsf{R}$, and $\mathcal{L}^*(e_1^k) = \mathsf{t\_EXITED}$,

Figure 20 shows the complete duration-aware process model obtained through `ShiftedDurR`(Task, MIN, MAX, ME, MAX_START, AlternativeActivity, RESET). Without loss of generality, the considered activity is represented as a task, i.e., $\alpha(a) = \textit{task}$ and $\mathcal{L}^*(a) = \mathsf{Task}$.

Figure 20 highlights that, compared to $\phi_{shifted}$, duration pattern $\phi_{shiftRes}$ includes another conditional event RESET, which leads back to ME in order wait for a new occurrence of the milestone event. We can assume that the condition associated to event RESET is *false* when the process begins and changes during execution, starting after the occurrence of milestone event ME.

8. Detecting and Managing Duration Violations

Whereas a first step towards the management of minimum and maximum duration constraints can be achieved by detecting constraint violations and by informing the process engine or the activity performer about potential problems/delays caused by temporal violations, appropriate constraint management deals with the specification of (temporal) exception handlers.

However, temporal exception management relies on the semantics and the nature of the violated constraints, and are strongly dependent upon the kind of activity being performed.

For instance, when a maximum duration constraint is violated, either sudden termination can be imposed or a side procedure can be used to expedite activity completion. Indeed, some activities need additional time to be correctly finished and they cannot be suddenly interrupted even if they violate maximum duration. As an example, we can think of a surgical intervention which is taking longer than planned: obviously, the surgeon cannot abandon the operating room, but additional workforce may be called to speed up the intervention.

Depending on their potential to interrupt activity execution, we distinguish between weak and strong maximum duration constraints. A *weak* duration constraint does not cause immediate activity interruption, but additional side activities can be performed with the goal of addressing constraint violation. Conversely, a *strong* duration constraint causes the sudden interruption of the on-going activity. For example, whenever drug therapy causes an unexpected allergic reaction, this must immediately stopped.

The introduced definition of constraint strength does not apply to minimum duration constraints, as there is no need for activity interruption in such context.

The process models proposed in the remainder of the section, are designed to manage duration violations at a high level of abstraction, since the nature of the repairing actions is highly dependent on the kind of task being performed.

To this end, we identify possible general and common actions that duration violation handlers are entitled to perform (see Table 6). Besides, we can also consider the following two extreme actions that hold for every constraint, that is, "Do nothing" (i.e., the handler simply alerts the activity performer that something went differently from expected) and "Terminate process execution".

Minimum Duration Violation	
Wait	The process waits before proceeding to the following element or a delay is added until minimum duration is observed.
Repeat	The whole activity or part of it is repeatedly executed until minimum duration is met.
Compensate	A compensation handler reverses the effects of the activity. In BPMN, compensation is used to revert the effects of a *Completed* activity that are no more desired [13].
Maximum Duration Violation (Weak)	
Escalate	Dedicated activities are performed to expedite completion. In BPMN, escalation identifies a situation that presumes some sort of reaction by the process and it is used to implement measures to expedite the completion of a process activity [13].
Extra Workforce	The activity is assigned to multiple resources that execute parts of it in parallel.
Maximum Duration Violation (Strong)	
Skip	The activity is interrupted and the remaining is skipped.
Undo	The activity is interrupted and its effects are reversed by some other activity.

Table 6: Possible behaviors of duration handlers, enacted in case of violation.

Regardless of which is the action taken to handle the violation, we assume that handlers are designed to resolve violations without violating other temporal constraints. That is, a minimum duration handler must resolve minimum duration violations without violating other temporal constraints (e.g., maximum duration constraints).

In [34], the importance of interrupting activities safely, i.e., by preserving their context, is discussed. Context preservation refers to the capability of the system to save data associated with the activity at the right time. In this direction, duration constraints can be seen as information related to activity execution, and thus, they must be dealt with when the activity is interrupted.

For both minimum and maximum duration violations, although we did not detail which activities are executed by the temporal exception handler, we adopt a modeling level of abstraction sufficient to ensure that handlers are correctly blended within the process.

8.1. Basic Process Models for Managing Duration Violations

In [11], we discussed different approaches for managing violations of simple duration constraints in a weak and a strong way. Let us start from the duration-aware process model depicted in Figure 3. Signal events t_minViolated and t_maxViolated detect duration violations and are used to trigger the corresponding violation handlers, that may have either a weak or a strong behavior.

For managing minimum duration violations caused by early activity interruption we used *compensation*, i.e., a way of undoing steps of a successfully completed activity whose results are no more desired and must be reversed. Minimum duration violations are managed by compensation handlers, that are constituted by a set of activities that are not connected to other portions of the BPMN model [13]. Instead, maximum duration violations are managed by event subprocesses. In BPMN, event subprocesses are a specialized kind of subprocess, that is included within a parent process but it is not part of the control flow. In other words, event subprocesses are *in-line handlers* triggered by events coming from the parent process [13].

In the remainder of this section, we consider the more complex case of activities having both interrupting and non-interrupting intermediate boundary events illustrated in Figure 8 and discuss how violation handlers defined in [11] have to be extended to deal with such a scenario.

When considering boundary events attached to a running activity, different interruption scenarios may occur. We begin with considering an activity having a weak maximum duration constraint, as depicted in Figure 21. We distinguish three different violation management behaviors depending on when Task is interrupted by the boundary event: (i) interruption occurs earlier than the minimum duration set for Task, (ii) interruption occurs within the expected duration range of Task or (iii) interruption occurs after the maximum duration limit set for Task.

For all cases (i), (ii) and (iii), it is necessary to distinguish the interruption of Task due to boundary event InterruptingE from regular task completion. To this end, a different signal t_INTERRUPT within exceptionFlow2 is added. This is

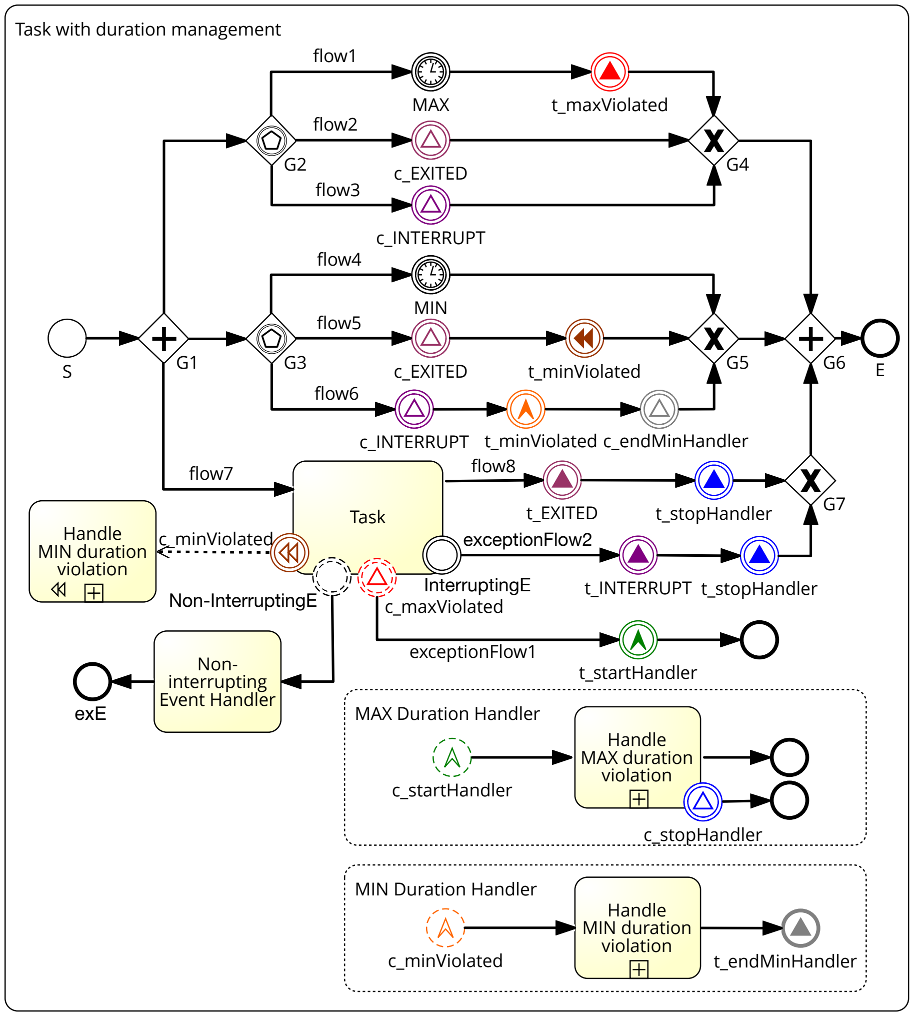


Figure 21: Process model for managing minimum and *weak* maximum duration constraints of activities having interrupting and non-interrupting intermediate boundary events.

essential to deal with minimum duration violation, as the activity can be either in state *Completed* or *Terminated*. The different name is chosen to highlight that the end of Task depends on causes other than anticipated completion.

(i) If InterruptingE occurs before the minimum duration set for Task, a temporal exception handling mechanism different from compensation must be enacted, as compensation only applies to successfully completed activities. For this reason, an event subprocess MIN Duration Handler is added to the parent process Task with duration management. Escalation event t_minViolated within flow6 is added to trigger the handler. Signal event c_endMinHandler waits for the event subprocess to complete, before letting the process flow proceed towards G5. Signal event t_stopHandler following signal t_INTERRUPT within exceptionFlow2 has no corresponding listening events on the process branches entitled of minimum duration management, as it is used for handling maximum durations.

(ii) If InterruptingE occurs within the expected duration range, signal event t_INTERRUPT is thrown to be caught by the corresponding event c_INTERRUPT, located on flow3. The signal behaves as previously explained for event c_EXITED, as the only difference is the state *Terminated* of Task. Again, signal t_stopHandler is thrown ineffectively, as no handler was initiated.

(iii) If InterruptingE occurs after MAX, as the activity is interrupted, we assume any possibly executing instance of subprocess MAX Duration Handler is also interrupted by signal event t_stopHandler. This interruption behavior is strong and, thus, it is in contrast with the notion of weak maximum constraint that has been addressed so far. However, as external occurrences presuppose strong interruption, there is no mean to preserve activity execution while the handler is operating on it.

When the event attached to Task boundary is non-interrupting, neither the management of minimum nor maximum duration constraints violations is affected. In case of weak maximum duration, we assume that the handler is designed to expedite Task completion and, thus, takes care of completing all the related non-interrupting event handlers attached to the activity.

The corresponding model for the management of violations of strong maximum duration constraints is reported in Figure 22.

In this setting, only two possible process behaviors are expected with respect to activity interruption: (i) the boundary event interrupts Task before MIN or (ii) at any moment within the desired time limits set for Task duration. Indeed, the interrupting nature of signal event c_maxViolated prevents any other interrupting event from occurring after MAX.

(i) If minimum Task duration is violated, signal event t_INTERRUPT is thrown. The different name is chosen to highlight that the end of Task depends on causes other than anticipated completion. As already explained, escalation event t_minViolated within flow6 triggers the corresponding event subprocess MIN Duration Handler.

(ii) If InterruptingE is triggered within the desired duration range, no duration violation handler is enabled. In this case, signal event t_INTERRUPT is thrown to be caught by the corresponding event c_INTERRUPT within flow3. Even if the final state of Task is *Terminated*, no further action is needed with respect to duration management.

9. Related Work

Temporal constraints specification and management has become a key aspect in the context of business process modeling and management, as constraining time is vital for reliable process actualization and execution [1, 2, 3].

Various proposals have addressed temporal constraint modeling within the research communities of workflows [1, 2, 3, 7, 8, 23, 35, 36, 37, 38] and BPM [5,

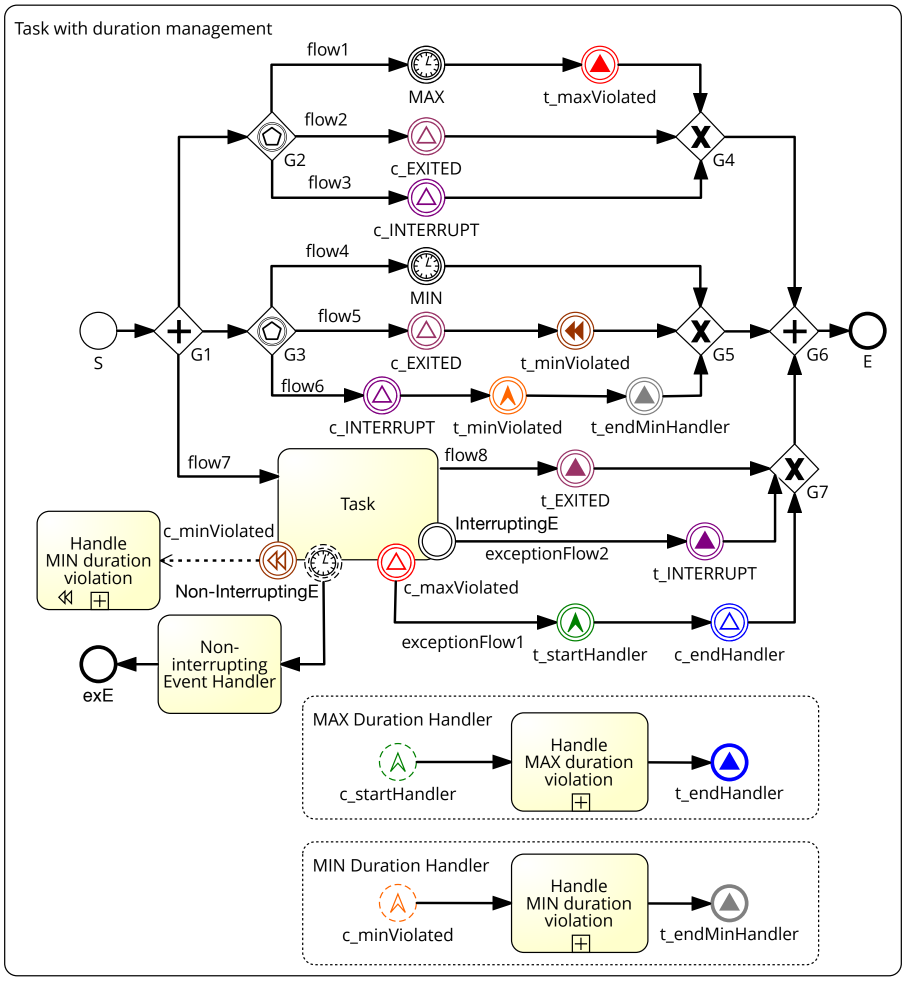


Figure 22: Process model for managing minimum and *strong* maximum duration constraints of activities having interrupting and non-interrupting intermediate boundary events.

11, 14, 15, 17, 39], drawing inspiration from approaches grounded in the formal principles of timed automata [16, 40], time Petri nets [41, 42], and temporal networks [24] for the verification of timed (process) models.

Time Petri nets are also used in the context of Web Service Composition for introducing temporal constraints. A model named H-Service-Net has been proposed in [43] to control and manage temporal consistency, and to support time constraints and exception handling.

In the remainder of the section, we describe selected relevant approaches belonging to the field of workflow modeling and, then, focus on proposals that specifically tackle temporal constraint modeling in BPMN [13]. Finally, we wrap up and compare our contribution with some of the discussed state-of-the art approaches, as detailed in Table 7.

9.1. Modeling and verification of temporal constraints in workflows

Early contributions to the representation of temporal information and temporal constraints are summarized in [1, 2]. The authors identify issues related to poor time consideration and reason on the effects that time violations have on business process costs and explain how proper time management may increase organizational competitiveness and improve timely reaction to external events and changes [10].

Focusing on handling globally distributed business processes, in [38] the authors underline the need of incorporating time-dependent factors, such as temporal order and time differences, into the logic of process activities at build-time. In detail, they consider activity duration and multiple time axes to represent different time zones, and define time constraints and process routing in terms of restrictions on starting and ending times of activities. Last but not least, a method for checking both the build-time and run-time consistency of the proposed time workflow model is presented.

In [7], the authors introduce a temporal model for conceptually designing clinical workflows, by addressing the representation of activity duration, delays, periodic and absolute constraints, and inter-activity constraints such as relative constraints and absolute delays. Verification of the modeled constraints is discussed and a Temporal Workflow Analyzer prototype for supporting workflow modeling and time management at design time is proposed. TNest, a new structured workflow language providing full support of temporal constraint specification during process design, is presented in [35]. The authors consider two main kinds of temporal constraints, namely activity durations and relative constraints, and tackle different nuances of the temporal workflow controllability concept, which was firstly introduced in [36, 37].

A major contribution to the formal specification and operational support of the temporal perspective in business processes is described in [3, 8, 23]. In [3], the authors identify a set of ten time patterns to ease the comparison of process-aware information systems (i.e., information systems that provide process support functions and separate the process logic from applications) and foster the choice of appropriate time constraints at design time. Furthermore, in [8] the authors provide an evaluation on the presented time patterns of selected process modeling approaches coming from both industry and academia, such as, among others, the BPMN [13] and BPEL standards, and those presented in [2, 7, 38].

Drawing inspiration from [8], when dealing with activity duration we do not consider the time span between the beginning of the activity and its real activation, that is we measure the time span between its start and end events. We also assume that sequence flows, gateways, and events consume a fixed or null amount of time and that any eventual data required to start the activity is available and its evaluation does not delay the execution. According to the evaluation introduced in [8], BPMN is said not to *directly* support minimum and range duration specification, whereas maximum duration can be partially supported by non-interrupting timer events attached to the activity's boundary. Additionally, it is highlighted that there is no mean to model the start time of a BPMN activity.

In [23], a formal temporal semantics is defined in order to avoid ambiguities and ease the practical use of time patterns proposed in [3, 8]. The authors exhaustively explore various aspects of the temporal perspective in business processes, by analyzing the identified patterns with respect to multiple design features and considering both process and time granularity. The studied temporal patterns have been extracted from a rich benchmark of business processes, mostly retrieved from the healthcare domain. BPMN 2.0 is found among the approaches that have systematically been reviewed with respect to temporal pattern support and expressiveness, as it will be later discussed. The authors summarize their previous efforts by providing a complete picture of their work: a formal semantics is discussed extensively for each time pattern and the ATAPIS Toolset is used for supporting the design, implementation, and verification of those temporal patterns mostly used in practice. ATAPIS is also used in [10] to implement change operations that allow modifying the temporal constraints of a time-aware process and to check the soundness of the changed processes.

Following the results presented in [23], we focused on evaluating the suitability of elements defined in the BPMN standard to represent duration constraints, by defining different processes called *duration patterns* and showing how to suitably combine them. Motivated by real case studies, we also considered the effects of external events on activity execution.

The definition, modeling, and management of temporal constraints encompasses the concept of *temporal constraints violation*. In [4], the authors propose an approach for managing controlled violations of time constraints in temporal workflows. Temporal constraints are expressed by means of formal expressions. However, not all constraints need to be strictly observed as the relaxation of some of them is allowed, provided that an associated penalty is applied.

Another aspect that is closely related to the one of temporal constraint violation is that of *predictable exception*, that is, a deviation that is known to possibly appear between what is planned and what is actually happening. In [44], the authors consider a particular kind of predictable exception, namely deadline escalation, which arises when an organization is not able to meet the deadlines for one or more instances of a process model. To improve the reaction to such predictable (temporal) exceptions, the authors propose a set of deadline escalation strategies to support decision-making and minimize tardiness.

Focusing on the modeling of duration constraints, this paper does not consider other important inter-activity temporal constraints, such as time-lags between activities and temporal loops [3, 5]. In addition, we do not deal with the formal verification of temporal constraints, as this would require using other approaches, such as those proposed in [23, 24, 30, 39, 45].

9.2. Expressing temporal and duration constraints in BPMN

The expressiveness of the Business Process Model and Notation [13] with respect to the modeling of temporal constraints has been addressed by some research proposals [5, 14, 15, 16, 24, 39, 40, 45, 46, 47, 48], most of which aimed to extend the standard in order to improve the specification and formal verification of temporal aspects.

Time-BPMN [15] is an extension proposed to capture the temporal perspective of business processes modeled with BPMN 1.2 [49], the previous version of the standard. The aim of the introduced extension is to simplify the representation of temporal constraints and dependencies that are vital for real business process enactment. Graphical markers are specified to control the start and end of activities and temporal constraint attributes are used to detail the targeted activity, the constraint kind, or additional useful documentation. Although activity start and finish times can be specified as inflexible constraints, the modeling of activity duration is not explicitly addressed. The weakness of BPMN to represent the temporal dimension of a process is compared to the expressiveness of project planning tools in [47]. In particular, the inability to visually represent the temporal execution order is highlighted, and the need for representing task duration is observed. BPMN process orchestration is analyzed and a preliminary representation of tasks with fixed duration is presented.

In [14], the authors extend BPMN 2.0 in order to enable the specification of temporal constraints and to verify potential violations by means of model checking approaches. In detail, the authors propose a graphical decorator depicting the minimum and maximum desired duration limits for an activity. Similarly to [15], constraint attributes are introduced to regulate the behavior and strength of the expressed duration constraints.

A BPMN extension designed to handle temporal constraints besides resource and concurrency ones is presented in [16]. The authors propose a mapping from BPMN elements to Timed Game Automata (TGA) to verify business processes and avoid design time and exceptions related to temporal and resource aspects. Flow objects are mapped onto timed automata, relations among them correspond to synchronization patterns between such automata, and process constraints are expressed as invariants, guards, and assignments on the timed automata. Regarding task duration modeling, attributes are added to BPMN tasks in order to explicitly specify their minimum and maximum execution times. A similar verification approach based on TGA is adopted also in [40] and applied to the set of temporal constraints introduced in [14].

Overall, the approaches introduced in [14, 16, 40] focus on translating BPMN into TGA for constraint verification. As a result, the modeling of temporal constraints in BPMN is mostly based on [15] and temporal aspects are expressed in terms of non-standard attributes and graphical decorators.

As for simple duration specification, the main difference with the proposals presented in [14, 16, 17] is the way we deal with duration specification. Designing dedicated processes for the management of duration violations allows us to specify the constraints in a more flexible way, that is, by allowing the activity to execute regardless of the defined constraints.

Instead, in the approach introduced in [17], graphical decorators are used to encode a "strong" duration semantics that enforces the activity to observe the constraints (both fixed and flexible durations), as interrupting boundary timer events are used for expressing duration. Moreover, it is not clear if and how duration of process regions may be expressed with the approaches presented in [14, 16, 17], especially for non-SESE regions. Only the concept of "combined duration" introduced in [4] may be close enough to express the duration of a process region.

Deferred duration and shifted duration are novel concepts, since in both cases duration depends on the occurrence of some event related to activity execution. In real-world processes, they are more typical of activities that unfold in time and are described at a quite high level of abstraction.

In [14] temporal constraints correlated with data constraints are exemplified by associating two different duration ranges to one activity, one of which is chosen depending on some data-based condition. However, the authors do not detail when the choice is made and what happens if conditions change while activity is being executed.

In [45] the authors propose an encoding of timed business processes into the Maude language, for automatically verifying some properties, yet considering a simpler extension of BPMN where only task timeouts and sequence flows delays can be expressed.

In [48], BPMN is extended for supporting Business Activity Monitoring, that is, the real-time monitoring and control capabilities during process runtime. Specifically, with respect to activity duration, a meta-model is proposed to measure the time span that goes from the moment the activity is assigned to a resource to the moment it is concluded.

Finally, a recent extension of BPMN considers raising the modeling of temporal control structures, such as temporal loops and temporal XOR-splits, at a conceptual level [5]. Temporal loops are associated to conditions that specify an upper temporal bound in addition to a regular loop-condition. That is, a loop iteration is executed if (i) the regular loop-condition holds and (ii) the time elapsed from the start of the process and a given time-point is less or equal than the specified upper temporal bound. Similarly, temporal XOR-splits compare the time elapsed from the start of the process and a given time-point with a constant and consequently route the process flow.

A novel relevant meta-model for the integration of temporal aspects in BPMN processes is presented in [17]. The authors provide a decorator-based

	Our approach	Modelling Processes with Time-Dependent Control Structures H. Pichler, J. Eder, M. Ciglic	Toward a Time-centric modeling of Business Processes in BPMN 2.0 S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel	Time-BPMN D. Gagné, A. Trudel	Formal Verification of Business Processes with Temporal and Resource Constraints K. Watahiki, F. Ishikawa, K. Hiraishi	A metamodel to integrate business processes time perspective in BPMN 2.0 C. Arevalo, M.J. Escalona, I. Ramos, M. Domínguez-Muñoz	Managing Decision Tasks and Events in Time-Aware Business Process Models R. Posenato, F. Zerbato, C. Combi	Temporal Specification of Business Processes through Project Planning Tools C. Flores, M. Sepúlveda
		[5]	[14]	[15]	[16]	[17]	[24]	[47]
Simple Duration Constraint of Activity	✓	✓	✓	✓	✓	✓	✓	✓
Simple Duration Constraint of Process Region	✓	-	-	-	-	-	-	-
Deferred Duration Constraint	✓	-	-	-	-	-	-	-
Shifted Duration Constraint	✓	-	-	-	-	-	-	-
Inter-Activity Temporal Constraint	-	-	✓	✓	-	✓	✓	✓
Temporal loops	-	✓	✓	-	-	✓	-	-
Temporal Constraint correlated with resource/data constraints	-	-	✓	-	✓	-	-	-
Definition of Semantics	✓	-	-	-	✓	✓	✓	-
Run-time Evaluation	✓	-	-	-	-	-	✓	-
Extended BPMN Notation	✓	✓	✓	✓	✓	✓	✓	✓
BPMN-compliant extension	✓	-	-	-	-	✓	-	✓
Formal Verification	-	✓	✓	-	✓	✓	✓	-

Table 7: Comparison between different approaches (✓: The considered approach supports the feature).

extension to the standard, where each constraint is expressed by using BPMN constructs. However, models proposed for activity duration are utterly different from those presented in this paper, mostly due to the intended semantics of event-based gateways [50]. Moreover, whereas the authors constrain activities to be executed within a fixed or flexible duration, we separate constraint specification from violation management and consider different levels of flexibility in both these aspects.

In [39], the authors propose a method to verify the controllability of time-aware business processes that consider constraints over activity duration. The approach suggests to specify both the structure and operational semantics of a process in terms of Constrained Horn Clauses (CHC). Then, also the notions of weak and strong controllability are encoded in CHC. Finally, two novel algorithms to solve the related strong and weak controllability problems are designed in order to deal with the computational cost given by nested universal and existential quantifiers.

Still in the context of controllability verification, in [24] the authors present time-aware BPMN processes and address dynamic controllability. In detail, they deal with adding temporal features to process elements, by considering the impact of events on temporal constraint management, by characterizing decisions with respect to when they are made and used within a process, by specifying and using two novel kinds of decisions based on how their outcomes are managed, and finally by considering intertwined temporal and decision aspects of time-aware BPMN processes to ensure proper execution.

Formal verification of time constraints is beyond the scope of this work. However, by guaranteeing that the modeled processes are sound (cf. Appendix A), we provide a way to evaluate constraints during the process run-time, by raising the due exceptions through signal events.

In Table 7, we summarize the comparison between this work and the discussed approaches, by considering both the kinds of addressed temporal constraints and the final research goal (e.g., run-time evaluation of temporal constraints, formal verification of temporal constraints).

10. Conclusions

In this work, we addressed the modeling of different duration constraints of activities and process regions, by using the BPMN standard. Structured and re-usable process models for specifying duration constraints at design time are proposed in Section 4–7 and they are enriched to provide detection and management of constraint violations at run-time as explained in Section 8. The different kinds of duration constraints and process models have been designed after studying real-world (clinical) settings, whose complex temporal aspects cannot be captured by simple duration constraints.

The main steps of our approach can be summarized as follows.

- We proposed a set BPMN ready-to-use duration-aware process models enclosing duration patterns for specifying different kinds of duration constraints, and for detecting possible constraint violations occurring at run-time. We entirely relied on the BPMN standard [13] for the definition of process semantics or on existing literature [18, 27, 23, 51, 52] whenever we considered that the semantics described in the BPMN standard was underspecified.

– We simulated the obtained processes with the Signavio Business Transformation Platform [53] to validate the behavior of the proposed models. We were able to conduct a step-by-step simulation as, to our knowledge, no current BPM software supports multiple-case simulation of processes having both event-based gateways and signal events. However, we used Signavio "step through simulation" to validate process behavior for all the cases introduced in Section 4 and Section 5. This kind of simulation allows one to decide which of the process elements ready for execution can be enabled. An example of step through simulation for the process model of Figure 3 is shown in Figure 23.

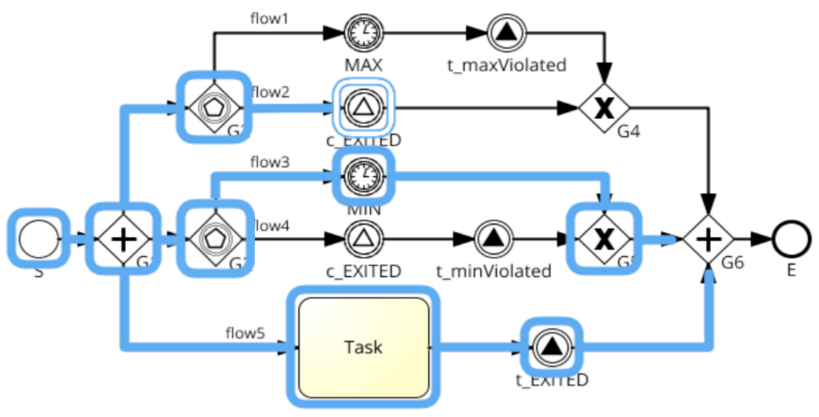


Figure 23: Example of step through simulation of the duration-aware process model of Figure 3. Active execution traces are highlighted.

– To validate the semantics of the designed process models in a more formal way, we manually derived an equivalent representation of the designed process models in terms of Time Petri Nets [42], considering both the mappings proposed in [27] and in [52], and results obtained on the soundness of *workflow nets*. The goal of this last step was to analyze the behavioral aspects of the derived nets to make sure that the specification of the operational semantics of the proposed process models was sound. Then, we also used both Romeo [54] and TINA (TIme Petri Net Analyzer) [55] for checking basic properties such as boundedness, liveness, and soundness of the obtained nets. In Appendix A, we describe the mapping of duration patterns onto time Petri nets and our validation approach.

For future work, we plan to consolidate our modeling approach by evaluating how the proposed patterns blend with complex real-world process models including also other kinds of temporal constraints and by simulating the obtained time-aware process models directly with BPM tools. Last but not least, we aim to deal with constraints applied to other modeling elements, first and foremost sequence flows, gateways, and events. In particular, we plan to investigate the possibility of having sequence flows that consume time and have the potential to delay activity activation.

References

[1] J. Eder, E. Panagos, M. Rabinovich, Workflow time management revisited, in: Seminal Contributions to Information Systems Engineering, Springer, Berlin, Heidelberg, 2013, pp. 207–213. doi:10.1007/978-3-642-36926-1_16.

[2] J. Eder, E. Panagos, M. Rabinovich, Time constraints in workflow systems, in: International Conference on Advanced Information Systems Engineering, Springer, 1999, pp. 286–300. doi:10.1007/3-540-48738-7_22.

[3] A. Lanz, B. Weber, M. Reichert, Workflow time patterns for process-aware information systems, in: Enterprise, Business-Process and Information Systems Modeling, Vol. 50 of Lecture Notes in Business Information Processing, Springer, Berlin, Heidelberg, 2010, pp. 94–107. doi:10.1007/978-3-642-13051-9_9.

[4] K. Akhil, R. Barton, Controlled violation of temporal process constraints – models, algorithms and results, Information Systems 64 (2017) 410 – 424. doi:10.1016/j.is.2016.06.003.

[5] H. Pichler, J. Eder, M. Ciglic, Modelling processes with time-dependent control structures, in: International Conference on Conceptual Modeling (ER), LNCS, Springer, Cham, 2017, pp. 50–58. doi:10.1007/978-3-319-69904-2_4.

[6] M. Weske, Business process management architectures, in: Business Process Management: Concepts, Languages, Architectures, Springer, Berlin, Heidelberg, 2012, pp. 333–371. doi:10.1007/978-3-642-28616-2.

[7] C. Combi, M. Gozzi, J. Juarez, B. Oliboni, G. Pozzi, Conceptual modeling of temporal clinical workflows, in: 14th International Symposium on Temporal Representation and Reasoning (TIME '07), IEEE, 2007, pp. 70–81. doi:10.1109/TIME.2007.45.

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

[9] M. Makni, S. Tata, M. Yeddes, N. Ben Hadj-Alouane, Satisfaction and coherence of deadline constraints in inter-organizational workflows, in: On the Move to Meaningful Internet Systems: OTM 2010, Springer Berlin Heidelberg, 2010, pp. 523–539. doi:10.1007/978-3-642-16934-2_39.

[10] A. Lanz, M. Reichert, Dealing with changes of time-aware processes, in: S. Sadiq, P. Soffer, H. Völzer (Eds.), Business Process Management, Springer International Publishing, Cham, 2014, pp. 217–233. doi:10.1007/978-3-319-10172-9_14.

[11] C. Combi, B. Oliboni, F. Zerbato, Modeling and handling duration constraints in BPMN 2.0, in: Proceedings of the 32nd Annual ACM Symposium on Applied Computing, SAC '17, ACM, New York, NY, USA, 2017, pp. 727 – 734. doi:10.1145/3019612.3019618.

[12] J. T. Daugirdas, Dialysis time, survival, and dose-targeting bias, Kidney international 83 (1) (2013) 9. doi:10.1038/ki.2012.365.

[13] Object Management Group, Business Process Model and Notation (BPMN), v2.0.2 (2014).

[14] S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, Toward a time-centric modeling of business processes in BPMN 2.0, in: Proceedings of International Conference on Information Integration and Web-based Applications & Services, IIWAS '13, ACM, New York, NY, USA, 2013, pp. 154–163. doi:10.1145/2539150.2539182.

[15] D. Gagne, A. Trudel, Time-BPMN, in: IEEE Conference on Commerce and Enterprise Computing, 2009, IEEE, 2009, pp. 361–367. doi:10.1109/CEC.2009.71.

[16] K. Watahiki, F. Ishikawa, K. Hiraishi, Formal verification of business processes with temporal and resource constraints, in: IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2011, IEEE, 2011, pp. 1173–1180. doi:10.1109/ICSMC.2011.6083857.

[17] C. Arevalo, M. J. Escalona, I. Ramos, M. Domínguez-Muñoz, A metamodel to integrate business processes time perspective in BPMN 2.0, Information and Software Technology 77 (Supplement C) (2016) 17–33. doi:10.1016/j.infsof.2016.05.004.

[18] F. Kossak, et al., A Rigorous Semantics for BPMN 2.0 Process Diagrams, Springer, 2014. doi:10.1007/978-3-319-09931-6.

[19] C. Combi, B. Oliboni, F. Zerbato, Towards dynamic duration constraints for therapy and monitoring tasks, in: Proceedings of the 16th Conference on Artificial Intelligence in Medicine, (AIME), Vol. 10259 of Lecture Notes in Computer Science, Springer, Cham, 2017, pp. 223–233. doi:10.1007/978-3-319-59758-4_25.

[20] M. Dumas, L. García-Bañuelos, A. Polyvyanyy, Unraveling unstructured process models, in: Business Process Modeling Notation: Second International Workshop, BPMN 2010, Potsdam, Germany, October 13-14, 2010. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 1–7. doi:10.1007/978-3-642-16298-5_1.

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

[22] P. Dadam, M. Reichert, K. Kuhn, Clinical Workflows — The Killer Application for Process-oriented Information Systems?, Springer London, London, 2000. doi:10.1007/978-1-4471-0761-3_3.

[23] A. Lanz, M. Reichert, B. Weber, Process time patterns: A formal foundation, Information Systems 57 (C) (2016) 38–68. doi:10.1016/j.is.2015.10.002.

[24] R. Posenato, F. Zerbato, C. Combi, Managing decision tasks and events in time-aware business process models, in: 16th International Conference on Business Process Management, BPM, Sydney, Australia, Sept. 9-14, 2018, Vol. 11080 of LNCS, Springer, 2018, pp. 102–118. doi:10.1007/978-3-319-98648-7_7.

[25] C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, R. T. Snodgrass, A glossary of temporal database concepts, ACM Sigmod Record 21 (3) (1992) 35–43. doi:10.1145/140979.140996.

[26] J. Mendling, H. A. Reijers, W. M. van der Aalst, Seven process modeling guidelines (7pmg), Information and Software Technology 52 (2) (2010) 127–136. doi:10.1016/j.infsof.2009.08.004.

[27] R. M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, Information and Software Technology 50 (2008) 1281–1294. doi:10.1016/j.infsof.2008.02.006.

[28] B. F. van Dongen, J. Mendling, W. M. van der Aalst, Structural patterns for soundness of business process models, in: 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06), IEEE, 2006, pp. 116–128. doi:10.1109/EDOC.2006.56.

[29] J. Dehnert, A. Zimmermann, On the suitability of correctness criteria for business process models, in: International Conference on Business Process Management, Springer, 2005, pp. 386–391. doi:10.1007/11538394_28.

[30] W. M. van der Aalst, Verification of workflow nets, in: Application and Theory of Petri Nets 1997: 18th International Conference, (ICATPN'97), Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 407–426. doi:10.1007/3-540-63139-9_48.

[31] M. Ciglic, Time management in workflows with loops, in: On the Move to Meaningful Internet Systems: OTM 2015 Workshops, Vol. 9416 of Lecture Notes in Computer Science, Springer International Publishing, Cham, 2015, pp. 5–9. doi:10.1007/978-3-319-26138-6_2.

[32] R. Johnson, D. Pearson, K. Pingali, The program structure tree: Computing control regions in linear time, in: ACM SigPlan Notices, Vol. 29, ACM, New York, NY, USA, 1994, pp. 171–185. doi:10.1145/773473.178258.

[33] L. Rangel-Castillo, S. Gopinath, C. S. Robertson, Management of intracranial hypertension, Neurologic clinics 26 (2) (2008) 521–541. doi:10.1016/j.ncl.2008.02.003.

[34] S. Bassil, S. Rinderle, R. Keller, P. Kropf, M. Reichert, Preserving the context of interrupted business process activities, in: Enterprise Information Systems VII, Springer Netherlands, Dordrecht, 2006, pp. 149–156. doi:10.1007/978-1-4020-5347-4_17.

[35] C. Combi, M. Gambini, S. Migliorini, R. Posenato, Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways, IEEE Transactions on Systems, Man, and Cybernetics: Systems 44 (9) (2014) 1182–1203. doi:10.1109/TSMC.2014.2300055.

[36] C. Combi, R. Posenato, Controllability in temporal conceptual workflow schemata, in: Proceedings of the 7th International Conference on Business Process Management (BPM 2009), 2009, pp. 64–79. doi:10.1007/978-3-642-03848-8_6.

[37] C. Combi, R. Posenato, Towards temporal controllabilities for workflow schemata, in: 17th International Symposium on Temporal Representation and Reasoning (TIME), 2010, pp. 129–136. doi:10.1109/TIME.2010.17.

[38] H. Zhuge, T. yat Cheung, H. keng Pung, A timed workflow process model, Journal of Systems and Software 55 (3) (2001) 231 – 243. doi:10.1016/S0164-1212(00)00073-X.

[39] E. De Angelis, F. Fioravanti, M. C. Meo, A. Pettorossi, M. Proietti, Verifying controllability of time-aware business processes, in: International Joint Conference on Rules and Reasoning, Springer, 2017, pp. 103–118. doi:10.1007/978-3-319-61252-2_8.

[40] S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, Enhancing formal specification and verification of temporal constraints in business processes, in: IEEE International Conference on Services Computing (SCC'14), 2014, pp. 701–708. doi:10.1109/SCC.2014.97.

[41] D. E. Saidouni, N. Belala, R. Boukharrou, A. C. Chaouche, A. Seraoui, A. Chachoua, Time petri nets with action duration: a true concurrency real-time model, International Journal of Embedded and Real-Time Communication Systems 4 (2) (2013) 62–83. doi:10.4018/jertcs.2013040104.

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

[43] F. Bey, S. Bouyakoub, A. Belkhir, Time-based web service composition, International journal on Semantic Web and information systems 14 (2) (2018) 113–137. doi:10.4018/IJSWIS.2018040106.

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

[45] F. Durán, G. Salaün, Verifying Timed BPMN Processes Using Maude, in: International Conference on Coordination Models and Languages, Springer, 2017, pp. 219–236. doi:10.1007/978-3-319-59746-1_12.

[46] S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, The temporal perspective in business process modeling: a survey and research challenges, Service Oriented Computing and Applications 9 (1) (2015) 75–85. doi:10.1007/s11761-014-0170-x.

[47] C. Flores, M. Sepúlveda, Temporal specification of business processes through project planning tools, in: Business Process Management Workshops: BPM 2010 International Workshops and Education Track, Springer, Berlin, Heidelberg, 2011, pp. 85–96. doi:10.1007/978-3-642-20511-8_8.

[48] J.-P. Friedenstab, C. Janiesch, M. Matzner, O. Muller, Extending BPMN for business activity monitoring, in: 2012 45th Hawaii International Conference on System Sciences, IEEE, 2012, pp. 4158–4167. doi:10.1109/HICSS.2012.276.

[49] Object Management Group, Business Process Model and Notation (BPMN), v1.2 (2009).

[50] F. Kossak, C. Illibauer, V. Geist, Event-based gateways: Open questions and inconsistencies, in: Business Process Model and Notation: Proceedings of the 4th International Workshop, (BPMN 2012), Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 53–67. doi:10.1007/978-3-642-33155-8_5.

[51] O. M. Group, BPMN 2.0 by example, version 1.0 non-normative (2010).

[52] M. Kunze, M. Weske, Business Process Models, 1st Edition, Springer International Publishing, Cham, 2016, pp. 125 –159. doi:10.1007/978-3-319-44960-9.

[53] Signavio GmbH, Signavio Business Transformation Suite (2017).

[54] D. Lime, O. H. Roux, C. Seidner, L.-M. Traonouez, Romeo: A parametric model-checker for petri nets with stopwatches, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, Berlin, Heidelberg, 2009, pp. 54–57. `doi:10.1007/978-3-642-00768-2\_6`.

[55] B. Berthomieu, F. Vernadat, Time petri nets analysis with TINA, in: Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems, QEST '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 123–124. `doi:10.1109/QEST.2006.56`.

[56] K. M. van Hee, N. Sidorova, J. M. van der Werf, Business Process Modeling Using Petri Nets, in: K. Jensen, W. M. van der Aalst, G. Balbo, M. Koutny, K. Wolf (Eds.), Transactions on Petri Nets and Other Models of Concurrency VII, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 116–161. `doi:10.1007/978-3-642-38143-0\_4`.

[57] A. H. ter Hofstede, W. M. van der Aalst, M. Adams, N. Russell, Modern Business Process Automation: YAWL and its support environment, Springer-Verlag Berlin Heidelberg, 2010. `doi:10.1007/978-3-642-03121-2`.

[58] W. M. van der Aalst, The application of Petri nets to workflow management, Journal of circuits, systems, and computers 8(1) (1998) 21–66. `doi:10.1.1.30.3125`.

[59] T. Murata, Petri nets: Properties, analysis and applications, Proceedings of the IEEE 77 (4) (1989) 541–580. `doi:10.1109/5.24143`.

[60] W. M. van der Aalst, K. M. van Hee, Workflow management: models, methods, and systems, MIT press, 2004.

[61] N. Lohmann, E. Verbeek, R. Dijkman, Petri Net Transformations for Business Processes — A Survey, in: K. Jensen, W. M. van der Aalst (Eds.), Transactions on Petri Nets and Other Models of Concurrency II, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 46–63. `doi:10.1007/978-3-642-00899-3\_3`.

[62] P. Merlin, D. Farber, Recoverability of communication protocols–implications of a theoretical study, IEEE transactions on Communications 24 (9) (1976) 1036–1043. `doi:10.1109/TCOM.1976.1093424`.

A. Validating Process Behavior by Means of Time Petri Nets

The goal of this Appendix is to provide an unambiguous operational semantics for the proposed BPMN processes by exploiting more formal time Petri nets to validate their behavior and prove their correctness.

Born to be understood by different users and to be employed in several organizational domains, process modeling languages often lack of a fully-specified, formal semantics. When a formal semantics exists, it is mostly defined in terms of transition systems or Petri nets [56]. Moreover, process engines such as YAWL [57] are based on Petri nets and, more specifically, on a special sub-class of Petri nets, called *workflow nets*.

In this context, the execution behavior of a process model can be specified in terms of Petri Nets, as their formal semantics is particularly suitable for disambiguating that one of BPMN and to check the correctness of process models [58]. In general, Petri Nets can be derived from BPMN processes by applying the mappings introduced in [27] and in [52], which differ from each other on the level of abstraction used to capture the life-cycle of a process activity.

In this paper, in order to be able to express the temporal dimension associated to BPMN activities and timer events, we adapt the mappings presented in [27, 52] to time Petri nets, that is, Petri nets with a possibly infinite time interval associated to each transition [42]. Thereby, we can capture the temporal aspects related to BPMN activities and timer events. Time Petri nets have been previously used in [9] for deadline constraints modeling. Last but not least, we ensure that the obtained time Petri nets conform to workflow nets, which are often used to model workflow systems.

The remainder of this appendix is structured as follows. Subsection A.1 introduces the basic theory of Petri Nets and time Petri nets. Subsection A.2 discusses the mapping of BPMN processes to (time) Petri nets. Finally, Subsection A.3 reports the validation results obtained by using time Petri net simulation and analysis software.

A.1. Petri nets: Basic concepts, notation, and semantics

Petri nets have been introduced as a mean to formally model concurrent systems, mostly under a control flow perspective. In BPM, they can be suitably applied to support process semantics specification, structural and behavioral property verification, and static analysis [58]. Petri nets token-based execution semantics makes them suitable for reproducing BPMN process execution, as its expressiveness is sufficient to reproduce the behavior of all the basic routing constructs of a BPMN process [59]. Besides, Petri nets are useful to provide an unambiguous graphical representation of business processes, suitable to prevent uncertainties and contradictions, which can easily arise when using other informal diagramming techniques [60].

A Petri net is a particular kind of directed, bipartite graph, formally defined as follows [59].

Definition A.1 (Petri Net). *A Petri net is a tuple $N = (P, T, A, M_0)$ where:*

- *P is the finite set of places, $P \neq \varnothing$;*
- *T is the finite set of transitions, $T \neq \varnothing$;*
- *$A \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs from places to transitions and from transitions to places;*
- *M_0 is the initial state (or marking) of the net, defined as a mapping $M_0 : P \rightarrow \mathbb{N}^+$ that assigns each place a nonnegative integer.*

With respect to the definition of Petri net provided in [59], we omit the weight function on the arcs, since we consider arcs to have a weight equal to 1.

Graphically, places are depicted as circles, whereas transitions are represented as rectangles. Moreover, it holds that $P \cap T = \emptyset$. A place $p \in P$ is called an *input place* for a transition $t \in T$ if and only if there exists a directed arc from p to t. Similarly, a place $p \in P$ is called an *output place* for a transition $t \in T$ if and only if there exists a directed arc from t to p.

Petri nets are traversed by tokens, drawn as black dots, that are collected by places and enable transitions. Transitions are the active components of a Petri net and they fire according to a defined *firing rule*. Specifically, a transition can fire when at least one token is available in each of its input places (i.e., the transition is *enabled*). When a transition fires, it removes one token from each of its input places and it adds one token to each of its output places.

The state M of a Petri net, sometimes called *marking*, is the distribution of tokens over places (i.e., $M \in P \rightarrow \mathbb{N}$). In this paper, we use the multiset notation to represent states, that is, $M = p1(n) \ldots p_i(n)$ denotes that place p_i contains n tokens. At any time, each place can have zero or more tokens and the number of tokens may change during the execution of the net. Given a Petri net N, a state M_n is said to be *reachable* from a state M_0 if there exists a firing sequence σ of transitions $t_0 \ldots t_n$, such that $M_0 \xrightarrow{\sigma} M_n$.

Petri nets have certain interesting properties that allow one to verify that the designed net behaves as expected. Here, we consider *liveness* and *boundedness* [30].

Definition A.2 (Liveness). *A Petri net $N = (P, T, A, M_0)$ is live if and only if, for every reachable state M_1 and every transition $t \in T$ there is a state M_2 reachable from M_1 which enables t.*

Definition A.3 (Boundedness). *A Petri net $N = (P, T, A, M_0)$ is bounded if and only if for every state reachable from M_0 the number of tokens in each place p does not exceed a finite number k.*

Whereas liveness ensures the complete absence of deadlocks, regardless of the chosen firing sequence, boundedness guarantees that there are no places in the net where tokens accumulate.

In general, structural constraints can be applied to Petri nets in order to better suit domain-specific goals and to promote the application of existing verification techniques.

In order to model business processes or workflow procedures, Petri nets must have some peculiar properties [30]. First of all, a Petri net must have a distinct source place (i.e., a single place that is not the target of any arc) and a distinct sink place (i.e., a single place that is not the source of any arc). Besides, all of its nodes must lie on some path from the source place to the sink place [61].

These structural restrictions identify an interesting sub-class of Petri nets, called workflow nets.

Definition A.4 (Workflow Net). *A workflow net is a tuple $WN = (P, T, A, M_0, e, c)$ where:*

- *(P, T, A, M_0) is a Petri net;*
- *e is the initial place;*
- *c is the final place.*

Nodes e and c are defined such that $e, c \in P$ or $e, c \in T$. All the other nodes of the net lie on a directed path from e to c.

Intuitively, a workflow net captures the execution of one instance of a business process, from its creation up to its completion.

The property of *soundness* is defined on top of workflow nets as follows.

Definition A.5 (Soundness). *A workflow net $WN = (P, T, A, M_0, e, c)$ is sound if and only if:*

- *(Safeness) each place cannot hold multiple tokens at the same time. Formally, $\forall p \in P,\ M(p) \leq 1$.*
- *(Option to complete) it is always possible to reach the state that marks the sink place c starting from the source place e and state M_0. Formally, $\forall_M (e \xrightarrow{*} M) \implies (M \xrightarrow{*} c)$;*
- *(Proper completion) state c is the only state reachable from state e with at least one token in place c. All other places must be empty. Formally, $\forall_M (e \xrightarrow{*} M \wedge M \geq c) \implies (M = c)$;*
- *(Absence of dead parts) for every transition t in the net, there is a sequence enabling it. Formally, $\forall_{t \in T}\ \exists_{M,M'}\ e \xrightarrow{*} M \xrightarrow{t} M'$.*

An interesting structural relation between workflow and Petri nets is reported below.

Theorem A.1. *If a Petri net has a source place e and a sink place c, by adding a transition t which connects the sink place to the source place we obtain a strongly connected net. Since every node $n \in P \cup T$ of the net now lies on a path from e to c, then the Petri net is a workflow net.*

The connection between the sink place and the source place is sometime called "short-circuit". As a result, the following property holds.

Theorem A.2. *A workflow net is sound if and only if the corresponding short-circuited Petri net is live and bounded.*

Moreover, well-structured process descriptions are guaranteed to be sound if they are live [29]. In this work, we consider workflow nets, as they result more intuitive in depicting the structure and execution steps of a business process.

However, in order to capture temporal aspects, we consider a particular class of high level Petri nets, i.e., *time Petri nets (TPNs).* TPNs have been introduced to support performance evaluation, safety determination, or behavioral properties verification in systems where time appears as a quantifiable and continuous parameter.

A time Petri net is a Petri net with a possibly infinite time interval associated to each transition [62]. The left extreme of the interval is the minimal time that must elapse from the time that all the input conditions of a transition are enabled until this can fire. The right extreme of the interval denotes the maximum time that the input conditions can be enabled and the transition does not fire. After this time, the transition must fire.

The formal definition of time Petri net, adapted from [42], is provided below.

Definition A.6 (Time Petri Net). *A time Petri net is defined as a tuple* $N = (P, T, A, M_0, I)$*, where:*
- *(P, T, A, M_0) is a Petri net;*
- $I : T \to \{\mathbb{R}^+, \mathbb{R}^+ \cup \{\infty\}\}$ *is a firing time function that associates an interval* $[\downarrow I(t), \uparrow I(t)]$*, called static firing interval, to each transition* t*.*

In line with this definition, each transition $t_i \in T$ has an associated time interval $[a, b]$, where $a \leq b$, and such that, once t_i has been enabled:
- a $(0 \leq a)$, is the minimum time that t_i must remain continuously enabled until it can fire;
- b $(0 \leq b \leq \infty)$, is the maximum time that t_i can remain continuously enabled without firing.

According to this scenario, any Petri net $N = (P, T, A, M_0)$ can be represented by an equivalent time Petri net, having an interval $[0, \infty]$ associated to each transition $t_i \in T$.

In the context of TPNs, the concept of state and of firing rule need to be revised. In particular, a state S of a time Petri net N is defined as the couple $S = (M, IS)$ consisting of:

(i) a marking M, which denotes the distribution of tokens in places and describes the logical part of the state;

(ii) a firing interval set IS, which is a vector of possible transition firing times and denotes the timed part of the state. The number of (ordered) elements of IS corresponds to the number of enabled transitions in M and each element $i \in IS$ is the time interval $[a, b]$ associated to enabled transition t_i.

According to the definition of time Petri net (cf. Def. A.6), transitions are *enabled* when every input place for a transition t_i has at least one token, as usually done in Petri nets.

However, an enabled transition t_i, associated to time interval $[a, b]$ cannot *fire* before a and later than b. Therefore, the relative firing time θ, which is related to the absolute enabling time ρ, must not be smaller than a of transition t_i and not greater than the smallest of the b's of all the transitions enabled by marking M (i.e., a of $t_i \leq \theta \leq \min\{b$'s of $t_k\}$, where k ranges over the set of transitions enabled in M).

As an example, let us consider the TPN depicted in Figure A.24, represented in state $S = (M_0, IS_0)$.

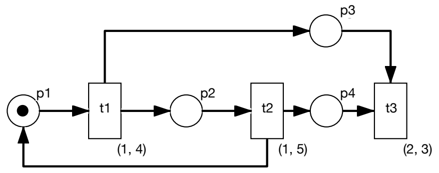


Figure A.24: Example of time Petri net, where $P = \{p1, p2, p3, p4\}$, $T = \{t1, t2, t3\}$ and initial marking $M_0 = p1(1),\ p2(0), p3(0), p4(0)$.

To show how the net in Figure A.24 can fire, we need to consider if both conditions (i) and (ii) hold for the current state S. Given a marking $M_0 = p1(1),\ p2(0), p3(0), p4(0)$, as $t1$ is the only *enabled* transition, $IS_0 = \{(1, 4)\}$, that is $1 \leq \theta_1 \leq 4$. Let us suppose that $\theta_1 = 2$, then

$$S \xrightarrow{(t1,\theta_1)} S' \begin{cases} M_0 = p1(1),\ p2(0), p3(0), p4(0) \xrightarrow{t1,\theta_1} M_1 = p1(0), p2(1), p3(1), p4(0) \\ IS_0 = \{(1, 4)\} \xrightarrow{t1,\theta_1} IS_1 = \{(1, 5)\} \end{cases}$$

This leads to state $S' = (M_1, IS_1)$, where $M_1 = p1(0),\ p2(1), p3(1), p4(0)$ and $IS_1 = \{(1, 5)\}$, where $t2$ is the only enabled transition. If $\theta_2 = 1$, we have that

$$S' \xrightarrow{(t2,\theta 2)} S'' \begin{cases} M_1 = p1(0), p2(1), p3(1), p4(0) \xrightarrow{t2,\theta 2} M_2 = p1(1), p2(0), p3(1), p4(1) \\ IS_1 = \{(1, 5)\} \xrightarrow{t2,\theta 2} IS_2 = \{(1, 4), (2, 3)\} \end{cases}$$

Now, we obtain $S'' = (M_2, IS_2)$, where $M_2 = p1(1),\ p2(0), p3(1), p4(1)$ and $IS_2 = \{(1, 4), (2, 3)\}$. In this case, both $t1$ and $t3$ are enabled: $t1$ can be fired for $\theta_1 \in (1, 3)$ and $t3$ can be fired for $\theta_3 \in (2, 3)$.

To summarize, below we show to possible alternative sequences of firings, one with $\theta_1 = 3$ and the other with $\theta_3 = 2$.

$$S'' \xrightarrow{(t1,\theta 1)} S''' \begin{cases} M_2 = p1(1), p2(0), p3(1), p4(1) \xrightarrow{t1,\theta 1} M_3 = p1(0), p2(1), p3(1), p4(1) \\ IS_2 = \{(1, 4), (2, 3)\} \xrightarrow{t1,\theta 1} IS_3 = \{(1, 5), (0, 0)\} \end{cases}$$

$$S'' \xrightarrow{(t3,\theta 3)} S'''' \begin{cases} M_2 = p1(1), p2(0), p3(1), p4(1) \xrightarrow{t3,\theta 3} M_4 = p1(1), p2(0), p3(0), p4(0) \\ IS_2 = \{(1, 4), (2, 3)\} \xrightarrow{t3,\theta 3} IS_4 = \{(0, 2)\} \end{cases}$$

A.2. Mapping BPMN processes to time Petri nets

Petri nets, and more specifically workflow nets, can be used to disambiguate the semantics of most BPMN core constructs and to statically check the semantic correctness of process models [58, 56]. In this paper, we use time Petri nets [42] to provide a formal foundation of the semantics of the proposed BPMN processes and to exploit existing tools for simulating and validating their execution.

In general, process activities can be modeled with Petri nets according to two different paradigms, that is, either by transitions or by places [56]. In the first case, the marking of the net indicates a situation of rest and transitions model activities that may lead to a new state of "rest". Oppositely, transitions can stand for instantaneous events and places may reflect a particular state of the net.

Within the BPM community, a widely applied mapping approach was proposed in [27]. An activity or an intermediate event is mapped onto a transition with one input place and one output place that models its execution, as shown in Figure A.25(a) for BPMN activity A.

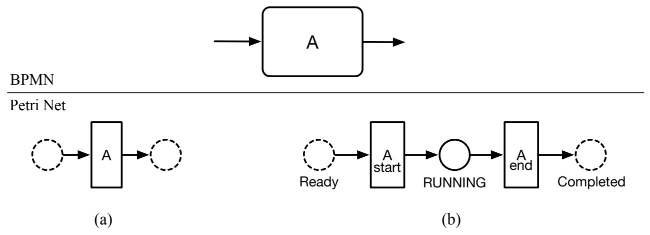


Figure A.25: Mapping of a BPMN process activity A onto a Petri net (a) abstracting from its life-cycle, as presented in [27], and (b) taking into account activity life-cycle, as introduced in [52]. Places drawn in dashed borders are shared with other net modules, as they are used as connecting elements.

A start or end event is mapped onto a silent transition, i.e., a transition whose firing cannot be observed, that signals when the process starts or ends. Gateways, except event-based and OR-split gateways, are also mapped onto silent transitions: AND-splits and AND-joins correspond to Petri net forks and joins, respectively, while BPMN XOR-split and XOR-joins correspond to Petri nets choice and return modules. In detail, data-driven exclusive gateways, are modeled as silent transitions having a common place as input and competing for a single token, so that the choice of which transition will fire is nondeterministic. For event-based gateways, the race condition between events is captured by having the corresponding transitions compete for tokens in the net place corresponding to the input flow of the gateway.

However, the mapping introduced in [27] does not consider the BPMN activity life-cycle (cf. Figure 1). In some circumstances, it is important to represent the different states that occur from the creation of an activity to its completion, as explained in [52]. To this end, each activity can be mapped to a net having three places corresponding to states Ready, RUNNING, and Completed, respectively. Transitions $\mathsf{A}_{\mathsf{start}}$ and $\mathsf{A}_{\mathsf{end}}$ denote the beginning and ending instants of the activity.

When a token is put on place Ready, the activity enters the state ready. This is properly represented by the firing rule of the Petri net: Transition $\mathsf{A}_{\mathsf{start}}$ can fire and, then, a token is removed from place Ready and put on place RUNNING. This token remains on place RUNNING, representing the execution of the activity until transition $\mathsf{A_{end}}$ fires, thus removing the token from place RUNNING and putting it on place Completed.

The mapping of BPMN activities to Petri nets by considering their life-cycle is shown in Figure A.25(b).

In this paper, we start from the mappings introduced in [27, 52] and map the proposed duration-aware processes onto time Petri nets in order to be able to express the temporal dimension associated to activities and timer events. In particular, we consider 1-bounded (or safe) TPNs having a delay-based semantics, as described in Section A.1. That is, the time interval associated to a transition expresses the minimum and maximum delays before an enabled transition can fire.

By applying the mapping in [27], we could model activities as transitions having one input and one output place, and associated to a positive time interval. However, when considering timed transitions, this mapping approach is imprecise, as the time interval associated to the transition represents the delay that exists from its enabling to its firing and, thus, it does not capture the duration of a running activity (i.e., the transition firing is instantaneous).

Therefore, we consider activity states in the obtained time Petri nets and model an activity A as a time Petri net, having three places (that correspond to states ready, running, and completing) and two (timed) transitions, whose firing capture the beginning and ending of the activity. In this way, activity duration, which is the time span that goes from the firing of $\mathsf{A_{start}}$ to the firing of $\mathsf{A_{end}}$, is captured by the time interval $[MIN, MAX]$ associated to the second transition that captures activity ending. Moreover, as we consider that activity duration does not cover the time span between the activation and the beginning of the activity, we consider transition $\mathsf{A_{start}}$ to be associated to a time interval of the form $[0, 0]$.

Similarly, timer events are modeled as transitions have one single input and one single output place and associated to a positive time interval. In detail, as we consider BPMN timer events that fire once a specified amount of time d has elapsed from their activation, the corresponding transition on the time Petri net will be associated to a time interval $[a, b]$ with $a, b = d$. Figure A.26 shows the mapping of BPMN activity A and timer event t onto the corresponding time Petri nets modules.

As for temporal aspects, all the remaining transitions in the net (i.e., transitions corresponding to gateways or other event types) are assumed to be associated to a time interval of the form $[0, \infty[$.

Finally, in order to model in Petri nets throwing and catching events that interact withing the same process, transitions corresponding to such events must be properly connected to each other. However, as the catching event may not be listening when the corresponding throwing event fires, deadlocks can occur or tokens may remain in the net, unconsumed. This is particularly true for signal events, due to their broadcast semantics [13, 18]. As unconsumed tokens prevent the net from completing properly, for each catching signal event in the BPMN process, two transitions are needed in the TPN: one that captures the fact that the event is caught, the other one that captures the situation in which

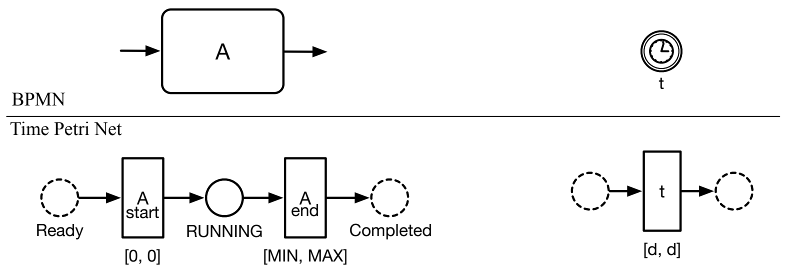


Figure A.26: Mapping of a BPMN process activity A and of a timer event t onto the corresponding time Petri nets. Places drawn in dashed borders are shared with other net modules, as they are used as connecting elements.

the event is not caught.

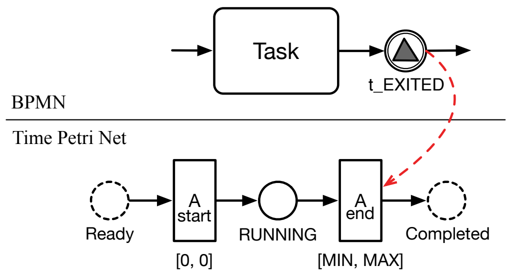


Figure A.27: Mapping of a BPMN task towards Petri net considering activity life-cycle [52]. Signal event t_EXITED is mapped to transition end_TASK.

Figure A.28 shows the time Petri Net obtained from the duration-aware process of Figure 3 as discussed above. As the meaning of signal event t_EXITED in the process is to detect Task completion, we map it directly to transition end_TASK, as shown in Figure A.27. The time interval [a, b] is written near each transition, but it is omitted for untimed transitions (i.e., when $[a, b] = [0, \inf[$). Transitions NOT c_EXITED capture the non-catching of event t_EXITED in the BPMN process, to guarantee proper completion of the net. As transitions c_EXITED are both in competition with transitions MIN and MAX (i.e., race condition of event-based gateway), the firing of MIN(MAX) must be related to the firing of NOT c_EXITED.

A.3. Process validation based on time Petri nets

In order to analyze the time Petri nets obtained from the proposed duration patterns and complete duration-aware processes and to check their soundness, we designed and verified the TPNs in Romeo [54] and TINA [55].

Both tools support various abstract state space constructions that preserve specific properties of state classes, such as absence of deadlocks, linear time temporal properties, or bisimilarity. Informally, a state class groups all the

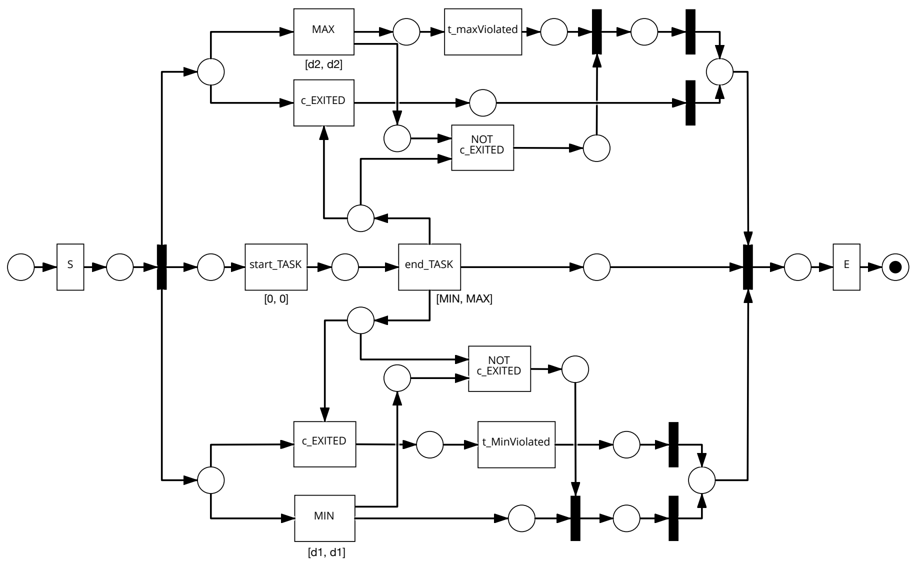


Figure A.28: Time Petri net obtained by applying the mapping proposed in [52] to the process of Figure 3, but considering the theory of time Petri nets [42].

states of a net according to all the possible firing times that are reachable from a given marking.

For instance, let us consider the net of Figure A.24 and its state $S'' = (M_2, IS_2)$, where $M_2 = p1(1)$, $p2(0), p3(1), p4(1)$ and $IS_2 = \{(1,4),(2,3)\}$. In this setting, state class $C2 = (M_2, D_2)$ is given by marking $M2$ and a domain D_2, which is the union of all firing intervals of the states, i.e., $D_2 = (1 \leq \theta 1 \leq 4 \cup 2 \leq \theta_3 \leq 3)$.

Besides working on state classes, both Romeo and TINA operate on standard time Petri nets. Therefore, in order to analyze our nets with respect to the soundness of workflow nets, we had to provide their short-circuited version. To do so, we connected the output place of transition E, with the input place of transition S by means of an additional transition. Figure A.29 shows the *stepper simulation* of the obtained short-circuited time Petri net in TINA. In the depicted scenario, transition end_TASK is expected to fire after transition MIN and transition MAX.

By assigning different time intervals to transitions end_TASK, MIN, and MAX, we derived multiple process execution traces that capture all the previously explained behaviors of the duration-aware process of Figure 3. By stepping through the net model in TINA and Romeo, we were able to observe that the time Petri nets reproduced the expected behavior of all the considered process traces. Romeo's simulation interface is shown in Figure A.31.

Finally, by analyzing the short-circuited TPN with TINA, we evinced that the net is live and bound, as reported in Figure A.30. Thus, thanks to theorem

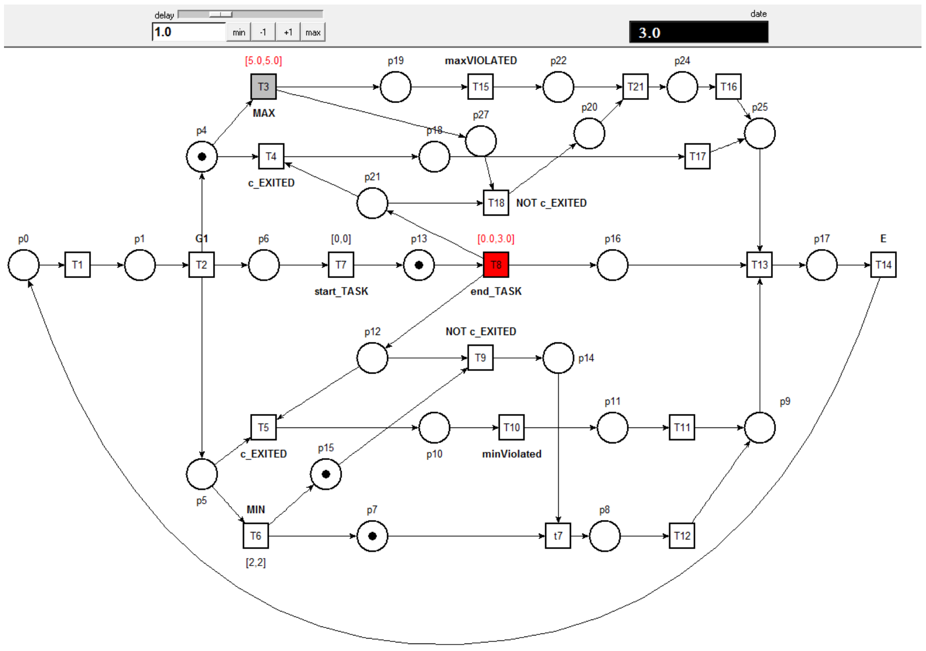
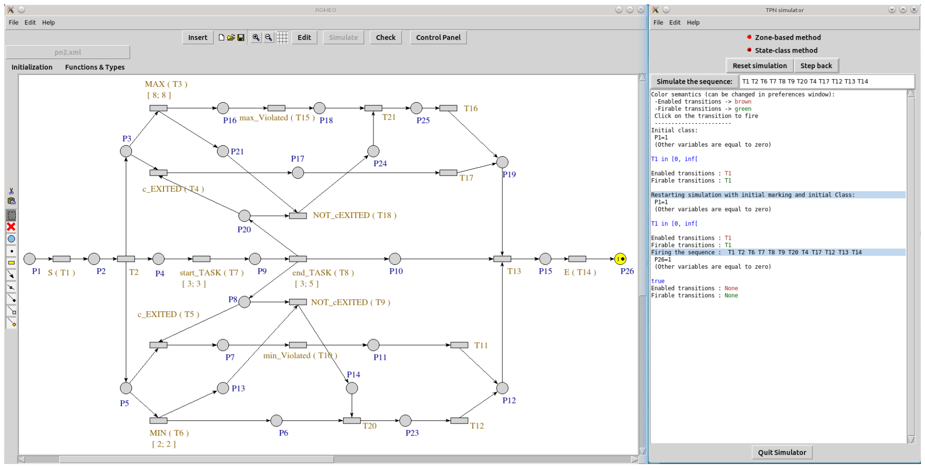


Figure A.29: Short-circuited time Petri net opened in TINA's stepper simulator. The delay represents the current relative delay, whereas the black bar on the top right reports the total execution time of the net. Transition end_TASK is enabled and can be fired within the next three time units.

A.2, we can state that the obtained TPN is sound.

```
LIVENESS ANALYSIS ----------------------------------------------
possibly live
possibly reversible

0 dead classe(s), 71 live classe(s)
0 dead transition(s), 20 live transition(s)

STRONG CONNECTED COMPONENTS:

0 : 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37
36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

SCC GRAPH:

0 -> T16/0, T12/0, T14/0, T21/0, T11/0, t7/0, T13/0, T15/0, T18/0, T10/0, T9/0, T17/0, T5/0, T6/0, T3/0,
T4/0, T8/0, T7/0, T2/0, T1/0

0.000s
ANALYSIS COMPLETED ---------------------------------------------

# net taskDuration, 24 places, 20 transitions                          #
# bounded, possibly live, possibly reversible                          #
# abstraction        count      props      psets       dead       live #
#      states           71         24          ?          0         71 #
# transitions          149         20          ?          0         20 #
```

Figure A.30: An excerpt of TINA's full textual output analysis results for the time Petri net shown in Figure A.28.



Simulate the sequence: T1 T2 T6 T7 T8 T9 T20 T4 T17 T12 T13 T14

```
Color semantics (can be changed in preferences window):
 -Enabled transitions -> brown
 -Firable transitions -> green
 Click on the transition to fire
 -----------------------
Initial class:
 P1=1
 (Other variables are equal to zero)

T1 in [0, inf[

Enabled transitions : T1
Firable transitions : T1

Restarting simulation with initial marking and initial Class:
 P1=1
 (Other variables are equal to zero)

T1 in [0, inf[

Enabled transitions : T1
Firable transitions : T1
Firing the sequence :  T1 T2 T6 T7 T8 T9 T20 T4 T17 T12 T13 T14
 P26=1
 (Other variables are equal to zero)

true
Enabled transitions : None
Firable transitions : None
```

Figure A.31: Simulating time Petri net with Romeo [54]. The simulated sequence is reported at the top right, and the *State-class method* [42] has been chosen for simulation. As done for TINA, the short-circuited net has been checked for absence of deadlocks.