

A Direct Formal Semantics for BPMN Time-Related Constructs

Sara Houhou, Souheib Baarir, Pascal Poizat, Philippe Quéinnec

▶ **To cite this version:**

Sara Houhou, Souheib Baarir, Pascal Poizat, Philippe Quéinnec. A Direct Formal Semantics for BPMN Time-Related Constructs. ENASE 2021 - 16th International Conference on Evaluation of Novel Approaches to Software Engineering, Apr 2021, online, Czech Republic. pp.138-149, 10.5220/0010462901380149 . hal-03170814

HAL Id: hal-03170814
https://hal.science/hal-03170814

Submitted on 16 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

A Direct Formal Semantics for BPMN Time-Related Constructs.

Sara Houhou[1,2,3] [a], Souheib Baarir[1,4], Pascal Poizat[1,4] [b] and Philippe Quéinnec[5]
[1]*Sorbonne Université, CNRS, LIP6, F-75005, Paris, France*
[2]*Biskra University, LINFI Laboratory, Biskra, Algeria*
[3]*Montpellier Université, CNRS, LIRMM, F-34000, Montpelier, France*
[4]*Université Paris Lumières, Université Paris Nanterre, F-92000, Nanterre, France*
[5]*IRIT - Université de Toulouse, F-31000 Toulouse, France*
{*sara.houhou, souheib.baarir, pascal.poizat*}*@lip6.fr, philippe.queinnec@irit.fr*

Keywords: BPMN, Timed Models, Workflows, Collaborations, Formal Verification, Alloy, Tool

Abstract: BPMN supports the design of intra-organization workflows and inter-organization collaborations. This rich notation includes elements to deal with models where time is central. However, the expressiveness of the BPMN time-related constructs hampers the definition of a formal semantics including them, and the provision of formal analysis means for timed process models. We propose here a first-order logic semantics for a subset of BPMN that includes its time-related constructs. With reference to related work, we support the specification of datetimes, durations, and cycles, using ISO-8601 formats as specified in the standard. Our approach is tool-supported by a model transformation into the Alloy formal language and its bounded counter-example generator. Our tool and model database are open source and freely available.

1 INTRODUCTION

A Business Process (BP) is a set of activities that are organized in order to reach some objective. Business Process Modeling Notation(BPMN) enables one to model both single processes using workflows but also collaborations where communication coordinates processes in different organizations. In both cases, time can play a part in the way the process(es) execute, *e.g.*, with deadlines before which one has to agree on a commercial offer or with sub-processes being regularly started after some duration.

To support this, BPMN defines a set of time-related events: timer start events (TSE) – to start some process, timer intermediary catch events ($TICE$) – to wait for some condition to be fulfilled, and timer boundary events (TBE), interrupting or not, to stop an activity or run a parallel one. All three kinds of events depend on a time-related condition defined in their *TimerEventDefinition* (OMG, 2013).

There are also three kinds of TimerEventDefinitions: *timeDate* (at some date-time, *e.g.*, at 5:30 pm on March, 24th, 2020), *timeDuration* (after some amount of time, *e.g.*, after 1 hour and 30 minutes), and *timeCycle* (repeated several times, with a duration between each instance and possibly after/before a given date-time, *e.g.*, after 1 hour and 30 minutes, 3 times, starting from 5:30 pm on March, 24th, 2020). To give the associated piece of information, BPMN relies on the ISO-8601 standard (ISO8601, 2004). Table 1 gives a synthetic view of time-related events in BPMN *w.r.t.* the ISO-8601 standard. No explicit formal semantics for BPMN time-related features is given in the standard. Further, the ISO-8601 description of time information is quite complex. This makes it more difficult to perform formal analysis of process models before, *e.g.*, running them on process engines.

In addition, a lot of effort has been done to identify the most common time-related scenarios from a business perspective, leading to *Process Time Patterns* (Lanz et al., 2010; Lanz et al., 2014; Lanz et al., 2016). Formally assessing the suitability of BPMN to support these patterns is desirable. However, this requires first to define a formal operational semantics for time-related features in BPMN. This issue has been addressed in several proposals (see Section 5). Still, these proposals often leave apart several features related to time-related events in the broader sense (Tab. 1).

Contribution. In a previous work (Houhou et al., 2019), we have defined a formal semantics for a sub-

[a] https://orcid.org/0000-0002-4166-0609
[b] https://orcid.org/0000-0001-7979-9510

Table 1: Time-related features in BPMN and their relation to ISO-8601, support: BPMN and us (•), BPMN only (○).

BPMN Standard			TSE	TICE	TBE Interrupt	TBE non-Interrupt
TimerEventDefinition	ISO-8601					
timeDate	date and time	yyyy-mm-ddThh:mm:ssZ	•	•	•	•
timeCycle	unbounded	R/ yyyy-mm-ddThh:mm:ssZ / yyyy-mm-ddThh:mm:ssZ	○	○	–	○
		R/ yyyy-mm-ddThh:mm:ssZ / PnYnMnDTnHnMnS	○	○	–	•
		R/ PnYnMnDTnHnMnS/yyyy-mm-ddThh:mm:ssZ	○	○	–	•
		R/PnYnMnDTnHnMnS	○	○	–	•
	bounded	Rn/ yyyy-mm-ddThh:mm:ssZ/yyyy-mm-ddThh:mm:ssZ	○	○	–	○
		Rn/ yyyy-mm-ddThh:mm:ssZ/PnYnMnDTnHnMnS	○	○	–	•
		Rn/ PnYnMnDTnHnMnS/yyyy-mm-ddThh:mm:ssZ	○	○	–	•
		Rn/ PnYnMnDTnHnMnS	○	○	–	•
timeDuration	duration	PnYnMnDTnHnMnS	–	•	•	•

set of BPMN (Fig. 1, but for time-related events), with a focus on BPMN communication constructs and different communication models. We also had defined a tool, fbpmn, to support the formal analysis of processes, and generate and animate counter examples. In the paper at hand, we extend this work by supporting the expressive time-related features of BPMN, including the use of the ISO-8601 standard, and show how we can support the Process Time Patterns. The proposed semantics is directly defined in terms of First-Order Logic (FOL), rather than through a mapping to some formal language. This choice makes it possible for researchers to map the FOL semantics to verification frameworks of their choice like (Alloy with SAT solvers) as we did here; TLA+ with the TLC model-checker (as we did before), SMT, or others). To support the formal verification of processes, we translate the proposed semantics in Alloy.

Case study. Fig. 2 presents a BPMN diagram of a simplified reviewing procedure for a scientific paper in a special issue of a journal. The scenario involves three participants that act in a collaboration diagram which are *PC chair, Author, Reviewer*. For simplification, we consider only one author and only one reviewer. This model is elaborated based on the following scope statements:

- The author sends a paper to the Journal PC Chair through the submit paper send task. Then, he/she will wait until the arrival of the notification response. If he/she doesn't receive a notification by "2021-09-18 T 00:00" the author withdraws his/her paper.
- The PC Chair starts when the specified date and time, "2021-01-17 T 00:00", of the CFP is reached. This is reflected by the timer start event of the process at the Journal PC Chair. Then, it waits for submissions. The receive activity is authorized until the specified close date, given as "2021-03-17 T 00:00". When the process receives a research paper, he/she assigns it to a reviewer via the send task "assign paper". To avoid delay for the response review process, he/she sends before the deadline date a reminder two times in a period of 15 days between. This is reflected by a non interrupting boundary time event associated with the receive review task, "R2/ P15D / 2020-04-29 T 00:00:00".
- The Reviewer process receives a Review Request message to starts. The reviewer starts preparing a review, and he/she sends it back to the PC Chair when it is ready.
- After the PC Chair has received a review, he/she prepares the acceptance/rejection letter, or a borderline letter if the paper requires further improvements. Then, he/she attaches the review to the notification letter and sends it to the author at the notification date and time specified in the CFP ("2021-05-16 T 00:00"). Even if the PC Chair has reached a decision before that date, he/she waits for this date before sending the notification. This is reflected by an intermediate timer event.

Overview. The remainder of this paper is structured as follows. Section 2 gives an informal overview of the BPMN execution semantics, including time-related features. Section 3 then details our proposed formalization for them. The implementation of our semantics in Alloy, verification, and evaluation are discussed in Section 4. We end up with a comparison to related work in Section 5 and a conclusions in Section 6.

2 OVERVIEW OF THE BPMN EXECUTION SEMANTICS

In this section we informally survey the semantics of the BPMN elements that we support (Fig. 1). To maintain traceability with the standard, we use a token-based approach, with tokens on both nodes and edges. We define an execution model based on two predicates, *Start* (*St*) and *Complete* (*Ct*). Their definition depends on the BPMN element taken in consideration. In the next section we give the formal

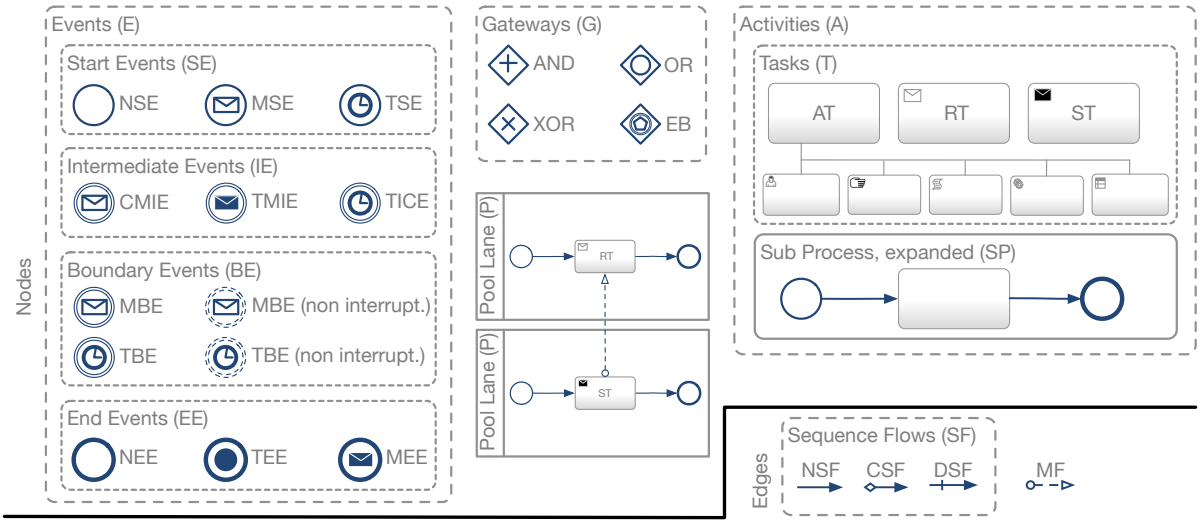


Figure 1: Subset of BPMN being taken into consideration in our work.

definition for time-related features, together with activities (tasks and sub-processes) and the event-based gateway, since time has an impact on their semantics. The semantics for the other BPMN constructs is kept from (Houhou et al., 2019).

Starting and terminating. Three kinds of events are used for the starting of a process[1]: none (*NSE*), message (*MSE*), and timer (*TSE*) start events. All three are defined only through completion predicates. They complete by initiating the process to which they belong and by generating a token on their outgoing edges. *MSE* and *TSE* are only ready to complete if (i) they are active and, respectively, (ii) a specified message has arrived on one of their incoming message flow edges, or the (global) clock has reached a given deadline. Two kinds of events are used for the termination of a (sub-)process: none (*NEE*) and terminate (*TEE*) end events. They start by moving a token from one of their incoming edges to themselves. A *TEE* has an additional behavior. It drops down all the remaining tokens of the process or sub-processes to which it belongs.

Activities. They are the main working units in a process. We make the distinction between a composite activity, or *Sub-Process* (*SP*), and an atomic activity, or *Task (T)*. The latter can be an abstract (*AT*), send (*ST*), or receive (*RT*) task. All activities have the same basic behavior. They start by moving a token from an incoming edge to themselves. When the activity is associated to a timer event, it does some additional work that we will specify in the next paragraphs.

Gateways. They are used to manage the control flow of a process. A gateway can be *Parallel* (*AND*) to model the simultaneous execution of all control flows in a set, *Inclusive* (*OR*) to model the simultaneous execution of a subset of control flows in a set, *Exclusive* (*XOR*) to model the choice of one control flow in a set, or *Event-Based* (*EB*) to model the choice of one control flow in a set, based on external event triggers (message or time conditions). An *EB* gateway is a kind of deferred choice since its completion depends on external events. It completes by moving the token it owns on the triggered event outgoing edge. In this paper we focus on *EB* gateways due to the importance of priority aspects in their formal execution semantics in presence of time-related features.

Intermediate events. They are events that can occur during the process execution. In our work, we consider the *Message* and *Timer* intermediate events. For the Message ones, we support message catching (*CMIE*), message throwing (*TMIE*), and boundary (interrupting or not) message (*MBE*) events. For the Timer ones, that constitute the focus of this work, we support the following:

- *Timer Intermediate Catch Events (TICE)* that act as a delay mechanism configured either by a duration ($TICE_p$[2]) or a fixed date ($TICE_d$). Such an event waits for the specified duration or date before letting the control flow on which it is located continue.
- *Timer Boundary Events (TBE)* that are attached to an activity. Such an event can be either interrupting ($TBE^{\oslash}$), *i.e.*, it interrupts the running of the activity it is attached to, or non-

[1] *NSE* can also be used to start a sub-process.

[2] In $TICE_p$, *p* stands for *period* as in the ISO-8601 standard for durations.

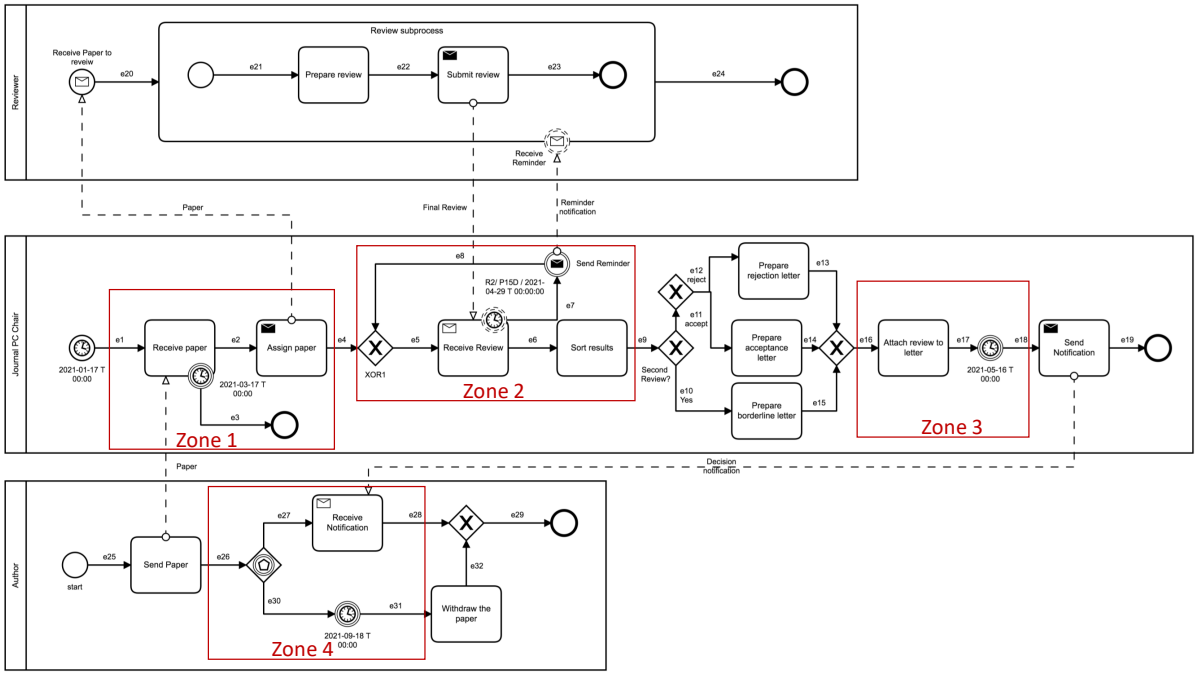


Figure 2: Paper Reviewing case study.

interrupting ($TBE^{\oplus}$). The start of an activity with timer boundary events causes the activation of local clocks for the boundary events attached to it.

- A $TBE^{\oslash}$ acts as a deadline for an activity. If the activation token remains on the activity more than a specific duration, or fixed date, the timer event interrupts the activity it is attached to. We separate $TBE^{\oslash}_{d}$ and $TBE^{\oslash}_{p}$: A $TBE^{\oslash}_{d}$ is a $TBE^{\oslash}$ configured with a date, while a $TBE^{\oslash}_{p}$ is configured with a duration.
- A $TBE^{\oplus}$ can be configured with a date ($TBE^{\oplus}_{d}$), a duration ($TBE^{\oplus}_{p}$), or a time cycle ($TBE^{\oplus}_{c}$). $TBE^{\oplus}_{d}$ and $TBE^{\oplus}_{p}$ define the same behavior as $TBE^{\oslash}_{d}$ and $TBE^{\oslash}_{p}$ (respectively), without cancelling the activity they are attached to. $TBE^{\oplus}_{c}$ might be triggered multiple times while the activity it is attached to is active. The number of cycles can either be fixed or unbounded. The time cycle definition associated to a timer event may have different configurations: (i) $TBE^{\oplus}_{c(start)}$ defines a number of recurrences for the timer event triggers separated by a period and where the first trigger is done relatively to a fixed start date; (ii) $TBE^{\oplus}_{c(end)}$ defines a number of recurrences for the timer event triggers separated by a period and where the last trigger is done before the fixed end date; (iii) $TBE^{\oplus}_{c(p)}$ defines a number of recurrences for the timer event triggers separated by a period.

3 BPMN TIME FORMALISATION

In this section we extend the formal semantics that we proposed in (Houhou et al., 2019), to handle time constructs with associated ISO-8601 time information. We rely on a (typed) graph representation of the workflow and collaboration models where types correspond to kinds of BPMN elements as given in Fig. 1 and Section 2, *e.g.*, TBE for Timer Boundary Events and $TBE^{\oslash}_{p}$ for interrupting Timer Boundary Events with a duration information. Roughly speaking, We consider two sets of basic elements types: Nodes, noted by T_{Nodes} and Edges, noted by T_{Edges} as follows:

- $T_{Nodes} = \{NSE, MSE, TSE, CMIE, TMIE, MBE, TSE, NEE, TEE, MEE, XOR, OR, EB, AND, AT, ST, RT, SP\} \cup TICE \cup TBE$, where:
  - $TICE = \{TICE_p, TICE_d\}$
  - $TBE = TBE^{\oslash} \cup TBE^{\oplus}$, with
    - $TBE^{\oslash} = \{TBE^{\oslash}_{d}, TBE^{\oslash}_{p}\}$
    - $TBE^{\oplus} = \{TBE^{\oplus}_{d}, TBE^{\oplus}_{p}\} \cup TBE^{\oplus}_{c}$, with $TBE^{\oplus}_{c} = \{TBE^{\oplus}_{c(start)}, TBE^{\oplus}_{c(start)}, TBE^{\oplus}_{c(p)}\}$
- $T_{Edges} = \{NSF, CSF, DSF, MF\}$

3.1 Syntax

A BPMN model is seen as a typed graph (Def. 1), where types corresponding to the BPMN elements are associated to nodes and edges. The BPMN standard defines three time categories: *timeDate*, that specifies a fixed date and time, *timeCycle*, that specifies repeating intervals, and *timeDuration*, that specifies the amount of time a timer should run before firing. Formally, we define three time categories, $Ctime = \{T_{date}, T_{duration}, T_{cycle}\}$, and the following time structures to characterize the time constraints, $timeVal = Date \cup Duration \cup Cycle$:

- $Date \subseteq \mathbb{N}$ represents a date (and time) expressed in seconds with respect to a reference date (1970-01-01T00:00:00Z). *Date* refers to the timeDate of the BPMN standard. A date like 2020-12-03T13:52:33Z in ISO-8601 format is converted to $1,607,003,553$ seconds.
- $Duration \subseteq \mathbb{N}$ represents a time duration in seconds. This corresponds to the timeDuration of the BPMN standard. A P3DT15M duration in ISO-8601 format (3 days and 15 minutes) is converted to $259,215$ seconds. Note that we do not support years and months in durations due to the ambiguity of their correspondence in seconds.
- $Cycle = (\mathbb{N} \cup \{\iota\}) \times [Duration \cup (Date \times Duration) \cup (Duration \times Date)]$, represents a composite timing type. It defines time redundancies along with a time duration, a fixed start date and time duration, or a time duration and a fixed end date. The number of repetitions is either bounded or not (ι).

Definition 1. *BPMN Graph. (extended from (Houhou et al., 2019)) Given T_{Nodes},T_{Edges}A BPMN graph is a tuple $\widehat{G}$= (N, E, $\mathbb{M}$, cat_N, cat_E, src, tgt, R, msg_t, attachedTo, isInterrupt, ftime) such that:*

- *N is the set of nodes,*
- *E ($N \cap E = \emptyset$) is the set of edges,*
- *$\mathbb{M}$ is the set of message types,*
- *$cat_N : N \to T_{Nodes}$ gives the type of a node,*
- *$cat_E : E \to T_{Edges}$ gives the type of an edge. In the following, we write N^T (resp. E^T) to denote the subset of nodes (resp. edges) of type T, e.g., $N^T = \{n \in N \mid cat_N(n) \in T\}$.By abuse of notation, in the following we write N^t instead of $N^{\{t\}}$, e.g., N^{NSE} instead of $N^{\{NSE\}}$, and similarly for E^t.*
- *$src/tgt : E \to N$ give the source/target of an edge,*
- *$R : N^{\{P,SP\}} \to 2^{N \cup E}$ gives the set of nodes and edges that are directly contained in a container (process or sub-process).*
- *$msg_t : E^{MF} \to \mathbb{M}$ gives the message associated to a message flow,*
- *$attachedTo : N^{BE} \to N^A$, gives the activity to which a boundary event node is attached,*
- *$isInterrupt : N^{BE} \to Bool$, denotes whether a boundary event node is interrupting or not,*
- *$ftime : N^{Timer} \to Ctime \times timeVal$, associates a time category and a value to the timer nodes.*

Several tools supports modelling with BPMN 2.0, e.g. Camunda, Eclipse modelling, Signavio…, and with various levels of completeness. These tools authorize modelling uncompleted models e.g., gateways without incoming and outgoing edges. To be analyzed, the BPMN models must be structurally well-formed. These well-formedness rules have been omitted due to lack of space.

Notation. R^+ is the transitive closure of R. To denote the projection of the function $ftime$ on a component of its co-domain, we use the notation $\lfloor_{Ctime}$(resp. $\lfloor_{timeVal}$): for example, if $ftime(n) = (T,V)$, where $T \in Ctime$ and $V \in timeVal$, then $ftime(n) \lfloor_{Ctime} = T$, and $ftime(n) \lfloor_{timeVal} = V$. Besides, when $ftime(n) \lfloor_{Ctime} = T_{cycle}$, then $ftime(n) \lfloor_{timeVal} = (r,d,p)$, with $(r,d,p) \in Cycle$. The projections $\lfloor_{timeVal_R}$, $\lfloor_{timeVal_P}$, and $\lfloor_{timeVal_D}$ give each element, with $ftime(n) \lfloor_{timeVal_R} = r$, $ftime(n) \lfloor_{timeVal_D} = d$ and $ftime(n) \lfloor_{timeVal_P} = p$.

Auxiliary functions. The following functions are defined: $in/out : N \to 2^E$ return the incoming/outgoing edges of a node, $in(n) = \{e \in E \mid tgt(e) = n\}$ and $out(n) = \{e \in E \mid src(e) = n\}$. A family of functions in^T (resp. out^T) : $N \to 2^E$ is used to combine *in* (resp. *out*) with E^T, $in^T(n) = in(n) \cap E^T$ and $out^T(n) = out(n) \cap E^T$. $procOf : N \to N^P$ returns the container process of a given node, $procOf(n) = p$ if and only if $n \in R^+(p)$.

3.2 Semantics

To describe a process execution, while maintaining traceability with the standard (OMG, 2013), we rely on a token-based semantics. We define an execution model using two predicates (St, Ct) for each node type. These correspond to, respectively, the enabling of a node to start its execution, and the enabling of a node to complete its execution. Some nodes only have a start transition (*e.g.*, end events), and others only have a completion transition (*e.g.*, gateways). The formal definition of these predicates relies on the notion of *state* of the BPMN Graph.

3.2.1 State Notion

It represents the global configuration of a BPMN model (workflow or collaboration), at any moment of its execution.

State. A state of a BPMN graph $\widehat{G}$= (N, E, $\mathbb{M}$, cat_N, cat_E, src,tgt, R, msg_t, $attachedTo$, $isInterrupt$, $ftime$) is a tuple $s = (m_n, m_e, l_c, g_c, rec)$ such that:

- $m_n : N \to \mathbb{N}$ and $m_e : E \to \mathbb{N}$, are marking functions, that associate a number of tokens to nodes and edges (respectively).
- $l_c : N^{Timer} \to \mathbb{N}$, is a local clock for the time spent on a timer node.
- $g_c \in \mathbb{N}$, is a global clock for the current time on the whole model.
- $rec : N^{TBE_c^{\oplus}} \to \mathbb{N} \cup \{\iota\}$ represents, for each activated non-interrupting timer boundary event node with a finite cycle, the number of occurrences that remains to be executed.

The set of all states of a BPMN graph is denoted by $States$.

Initial state. The initial state $s_o = (m_{n_0}, m_{e_0}, l_{c_0}, g_{c_0}, rec)$ of a BPMN graph is defined as follows:

- Nodes own 0 token, except for the initial nodes of a process that start with 1 token:

$$\forall n \in N, m_{n_0}(n) = \begin{cases} 1 \text{ if } \exists p \in N^P, n \in N^{SE} \cap R(p), \\ 0 \text{ otherwise} \end{cases}$$

- Edges own 0 token: $\forall e \in E, m_{e0}(e) = 0$;
- The global clock is initialized to a specific date and time (w.r.t. a referential[3]): $g_{c_0} \in \mathbb{N}$;
- Local clocks are initialized to zero: $\forall n \in N^{Timer}$, l_{c_0}(n)=0;
- Redundancy variables are initialized with the recurrence number (if it exists):

$$\forall n \in N^{TBE_c^{\oplus}}, \exists\, m,d,p \in \mathbb{N},\ ftime(n) \lfloor_{timeVal} = (m,p) \wedge ftime(n) \lfloor_{timeVal} = (m,d,p) \Rightarrow rec_0(n) = m$$

To formalize the semantics, we introduce the following predicates:

- $mayComplete(n) : N^{SP} \to Bool$, returns true if the sub-process may complete, *i.e.*, if there is a token on one of its end events, and there is no token on the rest of its elements (except for end events). This predicate is defined only for the sub-process where interruption nodes may attached to it.

$$\begin{aligned} \forall n \in N^{SP}, mayComplete(n) \stackrel{def}{\equiv}\ & (m_n(n) \geq 1) \\ & \wedge \forall e \in (R(n) \cap E), (m_e(e) = 0) \\ & \wedge \exists nn \in R(n) \cap N^{EE}, (m_n(nn) \geq 1) \\ & \wedge \forall x \in R(n) \cap (N \setminus N^{EE}), (m_n(x) = 0) \end{aligned}$$

- run is a predicate that increases the local clock of each active timer events node and the global clock at once.

$$run() \stackrel{def}{\equiv} \forall n \in S, (l'_c(n) = l_c(n)+1) \wedge (g'_c = g_c + 1)$$

- $\triangle$ is a predicate that denotes marking equality but for nodes and edges given as parameter, $\triangle(X)$ means "*nothing changes except for X*":

$$\begin{aligned} \triangle(X) \stackrel{def}{\equiv}\ & \forall n \in N \setminus X, m'_n(n) = m_n(n) \\ & \wedge \forall e \in E \setminus X, m'_e(e) = m_e(e) \end{aligned}$$

- $\triangle_t$ is a predicate that denotes that clocks do not change except for the local ones for the nodes in X:

$$\triangle_t(X) \stackrel{def}{\equiv} g'_c = g_c \wedge \forall n \in N^{Timer} \setminus X, l'_c(n) = l_c(n)$$

The formal semantics of the BPMN time-related constructs is given in Table 3. Here, we consider that m_n and m'_n (resp. m_e and m'_e) denote two successive markings of a node (resp. edge) in the execution semantics. The semantics of the elements in Fig. 1 and not in Table 3 is kept from (Houhou et al., 2019).

Let us consider a subset of timer nodes, called S, that groups the nodes that satisfy one of the following conditions: (i) all starting timer nodes that have a token and the global time date of the system does not reach their fixed time date (ii) all intermediate timer nodes that have an inactive local clock and have a marked incoming edge, or if they follow an event based gateway and the latter has a marked incoming edge, (iii) all boundary timer nodes attached to an active activity and their local clock is not active, or (vi) all active timer nodes (i.e., their local clocks are greater than 0 and they didn't reach their timing limits) as follows:

$$\begin{aligned} S \stackrel{def}{\equiv}\ & \{n \in N^{TSE} \mid (Timing(n) < g_c) \wedge (m_n(n) = 1)\} \\ & \cup \{n \in N^{TICE} \mid \exists e \in in^{SF}(n), (l_c(n) = 0) \wedge (m_e(e) > 0)\} \\ & \cup \{n \in N^{TICE} \mid \exists e \in in^{SF}(src(in^{SF}(n))), (l_c(n) = 0) \\ & \quad \wedge (m_e(e) > 0)\} \\ & \cup \{n \in N^{TBE} \mid (l_c(n) = 0) \wedge (m_n(attachedTo(n)) > 0)\} \\ & \cup \{n \in N^{Timer} \mid timing(n) > l_c(n) > 0\} \end{aligned}$$

[3] referential: an absolute date and time

Let Y be the subset of timer nodes in the BPMN graph that are ready to fire:

$$Y \stackrel{def}{\equiv} \{y \in N^{Timer} \mid l_c(y) \geq Timing(y)\}$$

To facilitate the reading of the transition relation, we define the following predicates:

- *step* defines a step of execution for a given node:

$$step(n) \stackrel{def}{\equiv} St(n) \vee Ct(n)$$

- *fztime* denotes time equality for the local clock of all timer nodes given as parameter:

$$fztime(Z) \stackrel{def}{\equiv} \forall z \in Z, l'_c(z) = l_c(z)$$

The transition relation distinguishes two cases. If at least a timer is ready to fire ($Y \neq \emptyset$), then a timer fires (it does a step) or an event-based gateway that precedes a firable timer does a step. Time does not advance, and other timers with the same expiration time can then fire. If no timer is ready to fire, all timers increase (*run*) and non-deterministically, a step can occur ($\exists n, step(n)$) or no step is done ($\triangle(\emptyset) \wedge \Xi$).

3.2.2 Transition Relation and Executions

The transition relation is a successor relation between states. It specifies that either a node makes a step (start or complete), or time advances but only if no timer node is ready to complete.

Transition Relation. Let s and s' be two states. We say that s' is a successor of s, iff the predicate $Next(s,s')$ (See equation. 1) holds.
States, here s and s', correspond to tuples of the form (m_e, m_n, l_c, g_c) and (m'_e, m'_n, l'_c, g'_c), whose elements are used in the definitions of *St* and *Ct*.

Execution. An execution is an infinite sequence of states such that $\sigma[0]$ is the initial state, and $\forall i \in \mathbb{N}, Next(\sigma[i], \sigma[i+1])$, where $\sigma[i]$ denotes the i^{th} state of the trace. Moreover, an execution is non-Zenon with regard to time and steps: there cannot be an infinite number of steps without time advancing, and there cannot be an infinite advancement of time without steps.

Formally, the non-Zenon hypothesis corresponds to weak-fairness on the left-hand part of the *Next* disjunction, and to weak-fairness on its right-hand part. This ensures that if one node is enabled, it will eventually be done.

4 Encoding of the Semantics in Alloy

In this section, we present how Alloy has been used to implement our FOL semantics and verify timed BPMN models.

4.1 Alloy

Alloy is a declarative modelling language based on FOL and relational calculus for expressing complex structural and behavioral constraints (Jackson, 2012). Alloy's logic is quite generic and does not commit to a particular specification style (Cunha, 2014). We believe that this is more natural and allows preserving the expressiveness of the input modelling language, BPMN in our case. Alloy comes with a tool, Alloy Analyser, a constraint solver that provides automatic simulation and verification based on a model-finding approach using a SAT solver. Alloy has been used for the verification of UML Activity Diagrams (Laurent et al., 2014), in the Model-Driven Engineering domain (Kleppe et al., 2003), for the modelling and analysis of distributed system protocols (Taghdiri and Jackson, 2003), networks (Georg et al., 2001), and safety and security concerns in critical systems (Dennis et al., 2004).

4.2 Implementation and Verification

Our semantics directly maps to Alloy which is also based on FOL. An element type (node or edge) is defined by an *abstract signature*, and the subtype relation relates these signatures (*e.g.*, $Node \supseteq Event \supseteq IntermediateEvent \supseteq TICE \supseteq TICE_d$). A BPMN graph inhabits these signatures with unique elements that correspond to the graph nodes and edges. Attributes mark the endpoints of an edge.

Listing 1: An excerpt of the implementation of our semantics in Alloy.

```
sig State {
 nodemarks : Node -> one Int,
 edgemarks : Edge -> one Int,
 network : set (Message -> Process -> Process),
 globalclock : one Int,
 localclock : (TimerStartEvent +
 TimerIntermediateEvent +
 TimerBoundaryEvent) -> one Int,
} // ...
pred completeTimerIntermediateEvent[s, s' : State,
n : TimerIntermediateEvent] {
  one ei : n.intype[SequentialFlow] {
    s.edgemarks[ei] > 0
    s.canfire[n]
    s'.edgemarks[ei] = s.edgemarks[ei].dec
    all eo : n.outtype[SequentialFlow]
        | s'.edgemarks[eo] = s.edgemarks[eo].inc
```

$$Next(s,s') \stackrel{def}{\equiv} (Y \neq \emptyset \wedge \left(\begin{array}{c} (\exists n \in Y : step(n) \wedge fztime(N^{Timer} \setminus \{n\})) \\ \vee \left(\begin{array}{c} \exists n \in N^{EB}, \exists eo \in out^{SF}(n), \\ (tgt(eo) \in Y) \wedge step(n) \wedge fztime(N^{Timer}) \end{array} \right) \end{array} \right) \\ \vee (Y = \emptyset \wedge run() \wedge fztime(N^{Timer} \setminus S) \wedge ((\exists n \in N : step(n)) \vee (\triangle(\emptyset) \wedge \Xi)) \quad (1)$$

```
delta[s, s', none, ei + n.outtype[SequentialFlow]]
deltaT[s, s', none]
  }                          }
pred State.canfire[n : TimerIntermediateEvent] {
{ n.mode in Date ∧
  (n.mode <: Date).date = this.globalclock
}
  or
{ n.mode in Duration ∧
this.localclock[n] = (n.mode <: Duration).duration
}                                                  }
```

Each semantic rule (Tab. 3) yields a *predicate*, syntactically identical to it (see Listing. 1 for *TICE*). As idiomatic in Alloy, an execution is an ordered set of *States*, where a *fact* (a constraint that always holds) relates two consecutive states (in this ordering). This fact is our predicate *Next*, a disjunction of the semantic rules and of time advance. An small excerpt is presented in Listing. 1. The resulting theories are available in the fbpmn repository (fbpmn, 2020) under `theories/alloy`.

Two kinds of verification are available, checks on the structure itself, and checks on the executions. For the first kind, *assertions* ensure that the model is well-formed, *e.g.*, a message flow connects two distinct processes. For the second kind, *predicates on States* are used to express properties on executions. We have defined :

- *Safe*, a predicate that states that no edge or node ever holds more than one token;
- *SimpleTermination*, a predicate that states that every process reaches a state where an End Event owns a token;
- *CorrectTermination*, a predicate that states that the whole system reaches a state where all processes have terminated with an End Event and no token are left on other nodes or edges.
- *MaxTime*, a predicate that states that the whole system reaches a final state before a given maximal time.
- *MinTime*, a predicate that states that the whole system takes at least a given minimal time to reach a final state.

Temporal constraints crossing the boundary of one process (e.g ., the deadline of message exchange), can be modelled in BPMN in two ways: (i) specify a latest date and time for the completion of a sending or receiving activity; this latter is modelled by associating a time boundary interrupting event to them (e.g, zone 1 in Fig.2). (ii) specify receiving actions on different control flow depending on time limitation; this latter is modelled using an event based gateway followed by a timer catch event (e.g, zone 4 in Fig.2)

Current experiments have allowed to validate our semantics on a subset of study cases models, and the implementation proved the feasibility of our approach, but unfortunately real-life models are often out of reach of Alloy Analyzer as the number of required states for an execution exceeds its capacity.

4.3 Application Rule Example

To perform the verification, the process and the properties are translated into an Alloy specification. Then, this specification is given as input to the Alloy Analyzer which reduces the verification to a SAT problem.

To explain our approach with an example, consider Zone 1 in Fig.2, an interrupting timer boundary event associated to the receive paper activity. As stated in Section.2, the start of an activity with timer boundary events causes the activation of the boundary events attached to it. If the specific fixed time-date defined on the boundary event is reached and the receive paper activity is still active, the boundary event interrupts the activity and generates a token on its outgoing edges. Fig. 3 shows a possible state which can be reached by the program. The configuration of this state is as follow: the PC chair process is active (R^{-1}(paperReview)=1), the global clock of the collaboration diagram reaches the time-date (2021-03-17 T 00:00), the receive activity review paper is active ($m_n(receivepaper)$= 1) and there is no paper message in the communication channel ($in^{MF}(receivepaper) = 0$). In this configuration; the interrupting timer boundary events completes, it drops the token from the receive paper activity ($m'_n(receivepaper)$= 0) and generates one on its outgoing edge ($m'_e(out^{SF}(tb1) = 1)$).

Listing 2 presents the alloy implementation of the Boundary timeDate rule. The latter corresponds to the First Order Logic formula (6) presents in Table 3.

Listing 2: An excerpt of the implementation of the semantics of boundary event rule in Alloy.

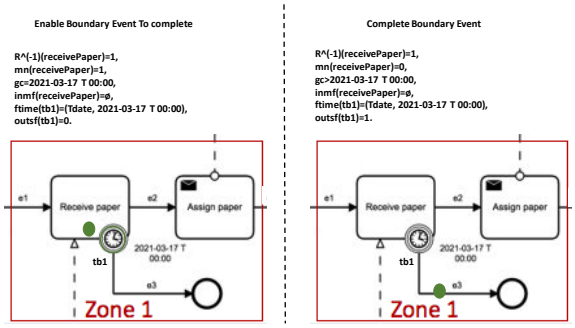


Figure 3: Completion of an Interrupting Timer Boundary Event.

```
/* Boundary */

pred State.cancompleteTimerBoundaryEvent[n : Node]
     ↪ {
    n in TimerBoundaryEvent
    this.nodemarks[n.attachedTo] > 0
    this.canfire[n <: TimerBoundaryEvent]
    (n.interrupting.isTrue ∧ n.attachedTo in
         ↪ SubProcess) ⟹ not this.
         ↪ cancompleteSubProcess[n.attachedTo]
}

pred completeTimerBoundaryEvent_Basic[s, s' :
     ↪ State, n : TimerBoundaryEvent, interrupted:
     ↪ lone Task] {
    s.nodemarks[n.attachedTo] > 0
    s.canfire[n]
    all eo : n.outtype[SequentialFlow] | s'.
         ↪ edgemarks[eo] = s.edgemarks[eo].inc
    (interrupted = none) or (s'.nodemarks[
         ↪ interrupted] = 0)
    s'.localclock[n] = 0  // actually only if
         ↪ Duration
    deltaN[s, s', interrupted, n.outtype[
         ↪ SequentialFlow]]
    deltaT[s, s', n]
}
```

5 RELATED WORK

Our paper focuses on the formalisation of BPMN time-related models that opens perspectives on quantitative analysis of business processes. This has been the subject of numerous works in the literature (Wong and Gibbons, 2009; Morales et al., 2010; Capel and Mendoza, 2012; Watahiki et al., 2011; Cheikhrouhou et al., 2013; Cheikhrouhou et al., 2014; Lanz et al., 2016; Combi et al., 2017; Combi et al., 2019; Durán and Salaün, 2017; Durán et al., 2018).

In (Wong and Gibbons, 2009), the authors define a time-related semantics for BPMN when using relative time constraints. The approach presents an abstract syntax for BPMN, based on the *Z* notation and a semantics based on *Communicating Sequential Processes* (CSP). It handles the particular case of time durations.

In (Morales et al., 2010; Capel and Mendoza, 2012), the authors propose an automated transformation from an extented version of BPMN 2.0 to timed CSP (CSP+T), as well as composition verification techniques for checking properties using the FDR2 model checker. They focus on the semantics proposed in (Wong and Gibbons, 2009).

In (Watahiki et al., 2011), the authors define a transformation approach for automatic generation of *timed automata* models from BPMN.

The works in (Cheikhrouhou et al., 2013),(Cheikhrouhou et al., 2014) address the need to model absolute and relative temporal constraints in timed automata. The authors extend BPMN models with a set of new notations to express temporal constraints, and then map timed business processes into *timed automata*.

In (Combi et al., 2017), the authors propose a formal specification of BPMN process diagrams based on *time Petri nets*. They model timer events of BPMN as transitions that have finite and positive temporal intervals. The authors represent a time duration as an interval, and they propose a mechanism for checking the duration constraints at run-time for the well-structured BPMN regions. They extend their work, in (Combi et al., 2019), by specifying a set of duration models using directly BPMN elements, without extension, to capture duration constraints for the activities.

The authors of (Durán and Salaün, 2017) present an approach for the formal verification and analysis of BPMN models. They propose an encoding of a subset of BPMN modelling elements into *Maude* where they introduce time duration and passing for tasks and flows. In this work they treat discrete (fixed) time duration. In (Durán et al., 2018), they extend their work by associating stochastic expressions to the tasks and flows in order to represent timing behaviors and they associate probabilities for triggering outgoing flows in case of choice (*i.e.*, exclusive and inclusive split gateways).

In (Lanz et al., 2014), the authors identify a set of time patterns to ease the comparison of process-aware information systems. The patterns are exhaustively analysed with respect to multiple design features. A formal semantics for the patterns, expressed as *temporal execution traces*, is given in (Lanz et al., 2016). They implement a set of time pattern variants in the ATAPIS toolset. The latter maps a process schema to *Conditional Simple Temporal Networks* in order to

Table 2: Comparison between approaches supporting BPMN time-constructs.

	reference	(Wong and Gibbons, 2009)	(Morales et al., 2010) (Capel and Mendoza, 2012)	(Watahiki et al., 2011)	(Cheikhrouhou et al., 2013) (Cheikhrouhou et al., 2014)	(Lanz et al., 2016)	(Combi et al., 2017) (Combi et al., 2019)	(Durán and Salaün, 2017) (Durán et al., 2018)	**ours**
	year	2008	2010-2012	2011	2013-2014	2016	2017-2019	2017-2018	2020
	formalism	CSP	CSP+T	Timed Aut.		Timed Ex. Traces	Time PN	Maude	FOL
BPMN	*TSE*	•	•	–	–	–	–	•	•
	TICE	•	•	•	•	–	•	•	•
	TBE non-interrupt	–	•	–	–	–	–	•	•
	TBE interrupt	•	•	•	•	–	–	•	•
time	timeDate	–	–	–	–	•	–	–	•
	timeCycle	–	–	–	–	–	–	–	•
	timeDuration	•	•	•	•	•	•	•	•
patterns	time lag	•	•	–	–	•	–	•	•
	duration	•	•	–	–	•	•	–	•
	fixed date element	–	–	–	–	•	–	–	•
	time dependent variability	–	•	–	–	•	–	–	•
	cycle element	–	–	–	–	•	–	–	•
	periodicity	–	–	–	–	•	–	–	•
other	time duration for activities	•	•	•	•	–	•	•	•
	time interval for edges	–	•	–	•	–	–	•	–

check the temporal consistency of process schema at design time, and check time violations at runtime.

Some works focus on extending the BPMN notation to capture temporal perspectives such as (Gagné and Trudel, 2009). This work presents a classification of flexible and inflexible temporal constraints (e.g. As Soon as Possible and As Late as Possible) and temporal dependencies between activities.

Discussion. The authors of (Lanz et al., 2010) define ten process time patterns suitable for evaluating the support of the temporal perspective in process-aware information systems (PAISs). According to the authors, BPMN supports six of these patterns. Table 2 gives a synthetic comparison between these proposals and ours. The table focuses on (1) covering all the time events in BPMN, considering their categories, and (2) showing how these works support the presentation of time patterns. As far as the BPMN coverage criteria is concerned, we can observe that we are among the approaches with a high coverage. A few studies address the evaluation of BPMN expressiveness with respect to its modeling elements, and most of them extend the notation in order to enhance the support of the standard towards time management constraints (Wong and Gibbons, 2009), (Morales et al., 2010; Capel and Mendoza, 2012), (Watahiki et al., 2011), (Durán and Salaün, 2017), (Durán et al., 2018).

As highlighted in Table 2, most of the existing works treat time duration for activities by extending BPMN by: (1) defining a non deterministic delay for a task (Wong and Gibbons, 2009) and (Watahiki et al., 2011) or, (2) representing a fixed duration a specified as an $[a,a]$ interval (Combi et al., 2017). However, BPMN gives the possibility to represent a duration for an activity using its own elements, without any extensions (see Fig. 4).

In addition, and to the best of our knowledge, no work in the literature allows one to specify the semantics for the different types of time information (*i.e.*, date, cycle, duration) associated to BPMN time-related events.

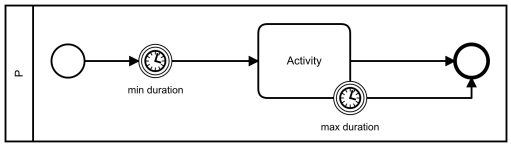


Figure 4: Specifying activity duration (minimum , maximum).

In this paper we cover the defined set of BPMN timer events in their full generality. As an example, consider the timer boundary event with cycle type. BPMN defines the cycle type with reference to ISO standard definition, where the ISO cycle type definition represents a complex construct which may be a repetition based on a duration until date or a repetition defined by a starting date and a period, or others (Section 2). To the best of our knowledge, most papers do not support all the variations of this construct (see Table 2). However, some works (Camunda examples, 2020), limited by the absence of a formalization, propose a simplified version of these events, e.g. every 10 minutes (a repetition on a defined period).

Note that, even if the work in (Lanz et al., 2014) provides a very rich formal semantics for process time pattern representation, it does not enable verification, and does not show their coverage *w.r.t.* standard BPMN elements.

6 CONCLUSION

We have proposed a formal semantics for the time-related constructs of BPMN. This semantics supports different combinations of events, time information categories (date-times, durations, cycles) and the corresponding ISO-8601 descriptions as prescribed by the BPMN standard. It can be then used to give a semantics to BPMN models using Process Time Patterns (Lanz et al., 2010; Lanz et al., 2014; Lanz et al., 2016). Our proposal is based on a direct formalization in First Order Logic which we have then implemented in the Alloy input language, together with an

extension of our fbpmn tool (Houhou et al., 2019) to transform BPMN models into Alloy modules.

The Alloy implementation of our semantics is direct. Yet, one still has to find a better representation of time steps in it in order to make automated verification amenable. We plan here to study the use SAT-based verification using Alloy Analyser in more case studies taking into account the execution time overage, max, min, waiting time synchronization, . . . ,etc. A longer term perspective concerns the support for data and conditional expressions over them through the use of a model checking modulo theories approach (Calvanese et al., 2019). Further, we plan to experiment with a direct implementation of the semantics proposed herein into the SMT-lib input language to solve the verification issue.

Acknowledgements. This work was supported by project PARDI ANR-16-CE25-0006.

REFERENCES

Calvanese, D., Ghilardi, S., Gianola, A., Montali, M., and Rivkin, A. (2019). Formal modeling and SMT-based parameterized verification of data-aware BPMN. In *BPM 2019*, pages 157–175.

Camunda examples (2020). BPMN 2.0 Symbol Reference. https://camunda.com/bpmn/examples/.

Capel, M. I. and Mendoza, L. E. (2012). Automating the transformation from BPMN models to CSP+ T specifications. In *35th IEEE Software Engineering Workshop (SEW)*, pages 100–109. IEEE.

Cheikhrouhou, S., Kallel, S., Guermouche, N., and Jmaiel, M. (2013). Toward a time-centric modeling of business processes in BPMN 2.0. In *15th IIWAS Conf.*, page 154.

Cheikhrouhou, S., Kallel, S., and Jmaiel, M. (2014). Toward a verification of time-centric business process models. In *23rd WETICE Conf.*, pages 326–331.

Combi, C., Oliboni, B., and Zerbato, F. (2017). Modeling and handling duration constraints in BPMN 2.0. In *Symposium on Applied Computing*, pages 727–734. ACM.

Combi, C., Oliboni, B., and Zerbato, F. (2019). A modular approach to the specification and management of time duration constraints in BPMN. *Inf. Syst.*, 84:111–144.

Cunha, A. (2014). Bounded model checking of temporal formulas with alloy. In *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 303–308.

Dennis, G., Seater, R., Rayside, D., and Jackson, D. (2004). Automating commutativity analysis at the design level. *SIGSOFT Softw. Eng. Notes*, 29(4):165–174.

Durán, F., Rocha, C., and Salaün, G. (2018). Stochastic analysis of BPMN with time in rewriting logic. *Sci. Comput. Program.*, 168:1–17.

Durán, F. and Salaün, G. (2017). Verifying Timed BPMN Processes Using Maude. In *COORDINATION 2017*, volume 10319 of *LNCS*, pages 219–236.

fbpmn (2020). fbpmn repository. https://github.com/pascalpoizat/fbpmn.

Gagné, D. and Trudel, A. (2009). Time-bpmn. In Hofreiter, B. and Werthner, H., editors, *2009 IEEE Conference on Commerce and Enterprise Computing, CEC 2009, Vienna, Austria, July 20-23, 2009*, pages 361–367. IEEE Computer Society.

Georg, G., Bieman, J., and France, R. (2001). Using Alloy and UML/OCL to specify run-time configuration management: A case study. In *pUML*, pages 128–141.

Houhou, S., Baarir, S., Poizat, P., and Quéinnec, P. (2019). A first-order logic semantics for communication-parametric BPMN collaborations. In *17th Int'l Conf. on Business Process Management*, volume 11675 of *LNCS*, pages 52–68.

ISO8601, I. (2004). ISO 8601:2004, data elements and interchange formats — information interchange — representation of dates and times. Standard, ISO.

Jackson, D. (2012). *Software Abstractions: Logic, Language, and Analysis*. MIT Press.

Kleppe, A. G., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co.

Lanz, A., Reichert, M., and Weber, B. (2016). Process time patterns: A formal foundation. *Inf. Syst.*, 57:38–68.

Lanz, A., Weber, B., and Reichert, M. (2010). Workflow time patterns for process-aware information systems. In *11th Workshop BPMDS*, volume 50 of *LNBIP*, pages 94–107.

Lanz, A., Weber, B., and Reichert, M. (2014). Time patterns for process-aware information systems. *Requir. Eng.*, 19(2):113–141.

Laurent, Y., Bendraou, R., Baarir, S., and Gervais, M.-P. (2014). Formalization of fUML: An application to process verification. In *CAiSE 2014*, pages 347–363.

Morales, L. E. M., Tuñón, M. I. C., and Pérez, M. A. (2010). A formalization proposal of timed bpmn for compositional verification of business processes. In *International Conference on Enterprise Information Systems*, pages 388–403. Springer.

OMG (2013). Business process modeling notation (BPMN). http://www.omg.org/spec/BPMN/2.0.2/.

Taghdiri, M. and Jackson, D. (2003). A lightweight formal analysis of a multicast key management scheme. In *FORTE 2003*, pages 240–256.

Watahiki, K., Ishikawa, F., and Hiraishi, K. (2011). Formal verification of business processes with temporal and resource constraints. In *SMC 2011*, pages 1173–1180.

Wong, P. Y. H. and Gibbons, J. (2009). A relative timed semantics for BPMN. *Electron. Notes Theor. Comput. Sci.*, 229(2):59–75.

APPENDIX

Table 3: An excerpt of BPMN semantic rules.

Rule	Node	Formulas
1	$n \in N^{TSE}$	$Ct(n) \overset{def}{\equiv} (ftime(n)\lfloor_{timeVal_D} = g_c) \wedge (m_n(n)=1) \wedge (m'_n(n)=m_n(n)-1)$ $\wedge \forall eo \in out^{SF}(n), (m'_e(eo)=m_e(eo)+1)$ $\wedge \exists p \in N^P, n \in R(p), (m_n(p)=0) \wedge (m'_n(p)=m_n(p)+1) \wedge \triangle(\{n,p\} \cup out^{SF}(n)) \wedge \triangle_t(\emptyset)$
2	$n \in N^{TICE_d}$	$Ct(n) \overset{def}{\equiv} \exists e \in in^{SF}(n), (m_e(e)=1) \wedge (ftime(n)\lfloor_{timeVal_D} = g_c) \wedge (m'_e(e)=m_e(e)-1)$ $\wedge (\forall e' \in out^{SF}(n), (m'_e(e')=m_e(e')+1)) \wedge \triangle(\{e\} \cup out^{SF}(n)) \wedge \triangle_t(\emptyset)$
3	$n \in N^{TICE_p}$	$St(n) \overset{def}{\equiv} \wedge (ftime(n)\lfloor_{timeVal_P} = l_c(n)) \wedge (l'_c(n)=0) \wedge \exists e \in in^{SF}(n), (m_e(e)=1) \wedge (m'_e(e)=m_e(e)-1)$ $\wedge \forall e' \in out^{SF}(n), (m'_e(e')=m_e(e')+1) \wedge \triangle(\{e\} \cup out^{SF}(n)) \wedge \triangle_t(\{n\})$
4	$n \in N^{TBE^{\oslash}_d}$	$St(n) \overset{def}{\equiv} \exists act \in N^{\mathbb{A}}, (act = attachedTo(n)) \wedge (m_n(act) \geq 1) \wedge (ftime(n)\lfloor_{timeVal_D} = g_c) \wedge isInterrupt(n)$ $\wedge ((act \notin N^{SP} \wedge m'_n(act)=0 \wedge (\forall ee \in out^{SF}(n), (m'_e(ee)=m_e(ee)+1))$ $\wedge \triangle(\{act\} \cup out^{SF}(n)) \wedge \triangle_t(\emptyset))$ $\vee (act \in N^{SP} \wedge \neg mayComplete(act) \wedge m'_n(act)=0$ $\wedge (\forall nn \in R(act) \cap N, (m'_n(nn)=0) \wedge (\forall ee \in R(act) \cap E, (m'_e(ee)=0)$ $\wedge (\forall out \in out^{SF}(n), (m'_e(out)=m_e(out)+1)) \wedge \triangle(\{act\} \cup R(act) \cup out^{SF}(n)) \wedge \triangle_t(\emptyset)))))$
5	$n \in N^{TBE^{\oslash}_p}$	$St(n) \overset{def}{\equiv} \exists act \in N^{\mathbb{A}}, (act = attachedTo(n)) \wedge (m_n(act) \geq 1) \wedge isInterrupt(n)$ $\wedge (ftime(n)\lfloor_{timeVal_P} = l_c(n)) \wedge (l'_c(n)=0))$ $\wedge ((act \notin N^{SP} \wedge (m'_n(act)=0) \wedge (\forall ee \in out^{SF}(n), (m'_e(ee)=m_e(ee)+1))$ $\wedge \triangle(\{act\} \cup out^{SF}(n)) \wedge \triangle_t(\{n\}))$ $\vee (act \in N^{SP} \wedge \neg mayComplete(act) \wedge m'_n(act)=0$ $\wedge (\forall nn \in R(act) \cap N, (m'_n(nn)=0) \wedge (\forall ee \in R(act) \cap E, (m'_e(ee)=0)$ $\wedge (\forall out \in out^{SF}(n), (m'_e(out)=m_e(out)+1)) \wedge \triangle(\{act\} \cup R(act) \cup out^{SF}(n)) \wedge \triangle_t(\{n\})))))$
6	$n \in N^{TBE^{\oplus}_d}$	$St(n) \overset{def}{\equiv} \exists act \in N^{\mathbb{A}}, (act = attachedTo(n)) \wedge (m_n(act) \geq 1) \wedge (ftime(n)\lfloor_{timeVal_D} = g_c) \wedge \neg isInterrupt(n)$ $\wedge (\forall out \in out^{SF}(n), (m'_e(out)=m_e(out)+1)) \wedge \triangle(\{act\} \cup R(act) \cup out^{SF}(n)) \wedge \triangle_t(\emptyset)))))$
7	$n \in N^{TBE^{\oplus}_p}$	$St(n) \overset{def}{\equiv} \exists act \in N^{\mathbb{A}}, (act = attachedTo(n)) \wedge (m_n(act) \geq 1) \wedge (\neg isInterrupt(n))$ $\wedge (ftime(n)\lfloor_{timeVal_P} = l_c(n)) \wedge l'_c(n)=0$ $\wedge (\forall out \in out^{SF}(n), (m'_e(out)=m_e(out)+1)) \wedge \triangle(\{act\} \cup R(act) \cup out^{SF}(n)) \wedge \triangle_t(\{n\})))))$
8	$n \in N^{TBE^{\oplus}_{c(start)}}$	$St(n) \overset{def}{\equiv} \exists act \in N^{\mathbb{A}}, act = attachedTo(n) \wedge (m_n(act) \geq 1) \wedge (\neg isInterrupt(n))$ $\wedge (ftime(n)\lfloor_{timeVal_D} = g_c) \wedge (rec(n) = ftime(n)\lfloor_{timeVal_R}) \wedge (l_c(n)=0)$ $\wedge (l'_c(n)=1) \wedge (rec'(n)=rec(n)-1)$ $\wedge (\forall ee \in out^{SF}(n), (m'_e(ee)=m_e(ee)+1)) \wedge \triangle(out^{SF}(n)) \wedge \triangle_t(\{n\}))$
		$Ct(n) \overset{def}{\equiv} \exists act \in N^{\mathbb{A}}, (act = attachedTo(n)) \wedge (m_n(act) \geq 1) \wedge (\neg isInterrupt(n))$ $\wedge (rec(n) \neq ftime(n)\lfloor_{timeVal_R}) \wedge (rec(n) \neq 0) \wedge (ftime(n)\lfloor_{timeVal_P} = l_c(n)) \wedge (l'_c(n)=1) \wedge (rec'(n)=rec(n)-1)$ $\wedge (\forall ee \in out^{SF}(n), (m'_e(ee)=m_e(ee)+1)) \wedge \triangle(out^{SF}(n)) \wedge \triangle_t(\{n\})$
9	$n \in N^{TBE^{\oplus}_{c(end)}}$	$St(n) \overset{def}{\equiv} \exists act \in N^{\mathbb{A}}, (act = attachedTo(n)) \wedge (m_n(act) \geq 1) \wedge (\neg isInterrupt(n)) \wedge (ftime(n)\lfloor_{timeVal_D} \neq g_c)$ $\wedge (ftime(n)\lfloor_{timeVal_P} = l_c(n)) \wedge (rec'(n)=rec(n)-1) \wedge (l'_c(n)=1)$ $\wedge (\forall ee \in out^{SF}(n), (m'_e(ee)=m_e(ee)+1)) \wedge \triangle(out^{SF}(n)) \wedge \triangle_t(\{n\}))$
10	$n \in N^{TBE^{\oplus}_{c(p)}}$	$St(n) \overset{def}{\equiv} \exists act \in N^{\mathbb{A}}, (act = attachedTo(n)) \wedge (m_n(act) \geq 1) \wedge (\neg isInterrupt(n))$ $\wedge (ftime(n)\lfloor_{timeVal_P} = l_c(n)) \wedge (rec'(n)=rec(n)-1) \wedge (l'_c(n)=1)$ $\wedge (\forall ee \in out^{SF}(n), (m'_e(ee)=m_e(ee)+1)) \wedge \triangle(out^{SF}(n)) \wedge \triangle_t(\{n\}))$
11	$n \in N^{EB}$	$Ct(n) \overset{def}{\equiv} \exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e)=m_e(e)-1)$ $\wedge \exists e' \in out^{SF}(n),\ tgt(e') \in N^{\{RT,CMIE\}} \wedge \exists mf \in in^{MF}(tgt(e')), (m_e(mf) \geq 1)$ $\vee tgt(e') \in N^{TICE_P} \wedge (l_c(tgt(e')) \geq ftime\lfloor_{timeVal_P}(tgt(e')))$ $\vee tgt(e') \in N^{TICE_d} \wedge (g_c \geq ftime\lfloor_{timeVal_D}(tgt(e')))$ $\wedge (m'_e(e')=m_e(e')+1) \wedge \triangle(\{e,e'\}) \wedge \triangle_t(\{\})$
12	$n \in N^{AT}$	$St(n) \overset{def}{\equiv} \exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e)=m_e(e)-1) \wedge (m'_n(n)=m_n(n)+1)$ $\wedge (\forall te \in N^{\{TBE^{\oplus}_p, TBE^{\oplus}_{c(p)}, TBE^{\oplus}_{c(end)}, TBE^{\oslash}_p\}}, (n = attachedTo(te)) \wedge (l_c(n)=0) \Rightarrow (l'_c(n)=1))$ $\wedge \triangle(\{n,e\}) \wedge \triangle_t(\{te \in N^{\{TBE^{\oplus}_p, TBE^{\oplus}_{c(p)}, TBE^{\oplus}_{c(end)}, TBE^{\oslash}_p\}} \mid (n = attachedTo(te))\})$
		$Ct(n) \overset{def}{\equiv} (m_n(n) \geq 1) \wedge (m'_n(n)=m_n(n)-1) \wedge (\forall e \in out^{SF}(n), (m'_e(e)=m_e(e)+1))$ $\wedge (\forall te \in N^{\{TBE^{\oplus}_p, TBE^{\oplus}_{c(p)}, TBE^{\oplus}_{c(end)}\}}, (n = attachedTo(te)) \Rightarrow (l'_c(n)=0))$ $\wedge \triangle(\{n\} \cup out^{SF}(n)) \wedge \triangle_t(\{te \in N^{\{TBE^{\oplus}_p, TBE^{\oplus}_{c(p)}, TBE^{\oplus}_{c(end)}\}} \mid (n = attachedTo(te))\})$
13	$n \in N^{SP}$	$St(n) \overset{def}{\equiv} \exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e)=m_e(e)-1) \wedge (m'_n(n)=m_n(n)+1)$ $\wedge (\forall n_{se} \in (N^{TSE} \cap R(n)), (m'_n(n_{se})=m_n(n_{se})+1))$ $\wedge (\forall te \in N^{\{TBE^{\oplus}_p, TBE^{\oplus}_{c(p)}, TBE^{\oplus}_{c(end)}, TBE^{\oslash}_p\}}, (n = attachedTo(te)) \Rightarrow (l'_c(n)=1))$ $\wedge \triangle(\{e,n\} \cup (N^{TSE} \cap R(n))) \wedge \triangle_t(\{te \in N^{\{TBE^{\oplus}_p, TBE^{\oplus}_{c(p)}, TBE^{\oplus}_{c(end)}, TBE^{\oslash}_p\}} \mid (n = attachedTo(te))\})$
		$Ct(n) \overset{def}{\equiv} (m_n(n) \geq 1) \wedge (m'_n(n)=0)$ $\wedge (\forall e \in R(n) \cap E, (m_e(e)=0))$ $\wedge (\exists n_{ee} \in (N^{EE} \cap R(n)), (m_n(n_{ee}) \geq 1)) \wedge (\forall nn \in R(n) \cap N, (m_n(nn) \geq 1 \Rightarrow nn \in N^{EE}))$ $\wedge (\forall nn \in (R(n) \cap N^{EE}), (m'_n(nn)=0)) \wedge (\forall e \in out^{SF}(n), (m'_e(e)=m_e(e)+1))$ $\wedge (\forall te \in N^{\{TBE^{\oplus}_p, TBE^{\oplus}_{c(p)}, TBE^{\oplus}_{c(end)}\}}, (n = attachedTo(te)) \Rightarrow (l'_c(te)=0))$ $\wedge \triangle(\{n\} \cup (R(n) \cap N^{EE}) \cup out^{SF}(n)) \wedge \triangle_t(\{te \in N^{\{TBE^{\oplus}_p, TBE^{\oplus}_{c(p)}, TBE^{\oplus}_{c(end)}\}} \mid (n = attachedTo(te))\})$