A Precise Execution Semantics for BPMN

Vitus S.W. Lam

***Abstract*—Bringing a high confidence to the validity of business processes is one of the prevailing themes in business process management. Regardless of the introduction of BPMN 1.2, there is no discernible improvement in the preciseness of the BPMN semantics. Motivated by the need to ensure the trustworthiness of BPMN models and the absence of an accurate behavioural semantics, a rigorous semantic definition of BPMN is advocated. The symbolic encodings of the execution semantics are expressed concisely using linear temporal logic (LTL). The LTL-based representations serve as a basis for the formal analysis of BPMN diagrams as well as the prototypical implementation of software tools.**

***Index Terms*—BPMN, execution semantics, behavioural semantics, linear temporal logic.**

I. Introduction

The business process modelling notation (BPMN) [1], [2], [3] is emerging as a widely accepted approach in the domain of business process management. The BPMN, which embodies a collection of notational elements, is a visual modelling language for the construction of business process diagrams. In the BPMN specifications [1], [3], the behavioural semantics is elucidated using the notion of token flow. The Petri-like semantics of BPMN is along the same vein as the one in UML activity diagrams [4].

While the syntax of BPMN is specified by the official specifications in a precise manner, the execution semantics of BPMN is solely described in narrative form using plain text. Since a rigorous semantics is a prerequisite for the verification of a BPMN model, we take on the challenge of defining the behaviour of BPMN elements in the form of linear temporal logic [5], [6]. The rationale behind the adoption of linear temporal logic in lieu of computation tree logic is due to the fact that the BPMN semantics matches closely with a linear time model.

The rest of the paper proceeds as follows. Section 2 reviews prior work in the area. Section 3 is dedicated to an overview of BPMN. The syntax of linear temporal logic is presented in Section 4. Section 5 seeks to provide a semantic foundation for BPMN on the basis of linear temporal logic. The application of the formalized execution semantics is exemplified in Section 6. Section 7 offers brief concluding comments as well as points to ideas for future research.

II. Related Work

In the business process management community, there is a growing body of literature regarding BPMN. Most of these contributions emphasize the static analysis of BPMN models and can be classified as four main approaches. The first approach exploits Petri nets as the underlying framework for assessing the behavioural correctness of BPMN models. Raedts et al. [7] adopt Petri nets as an intermediate representation when transforming BPMN models into mCRL2 for performing formal analysis. Dijkman et al. [8] propose the utilization of ProM framework for studying the behaviour of BPMN models by defining a semantic mapping between BPMN and Petri nets. To verify BPMN models against different properties by means of CPN Tools [9], Ou-Yang and Lin [10] propound an approach that BPMN models are first translated into BPEL4WS and then encoded as Colored Petri-net XML (CPNXML).

Both the second and third approaches that are discussed in the following, unlike the first approach, harness the power of process calculi. They utilize the π-calculus [11], [12] and Communicating Sequential Processes (CSP) [13] as the semantic formalisms, respectively. Bog et al. [14], [15], [16], [17] put forward the simulation and analysis of BPMN models using PiVizTool by translating BPMN into the π-calculus. The work of Puhlmann [18] formalizes BPMN in the form of the π-calculus and verifies the respective π-calculus representations through the use of Advanced Bisimulation Checker (ABC) [19]. Wong and Gibbons [20], [21] convert BPMN into CSP in order to determine the compatibility between BPMN processes with the Failures-divergence Refinement (FDR) model checker [22]. In [23], [24], Wong and Gibbons develop their idea a little further by establishing a relative-time semantic model built upon CSP. The application of symbolic model checking on the verification of BPMN models constitutes the fourth approach. A semantic model for BPMN 1.0 is defined using New Symbolic Model Verifier (NuSMV) language [25] in [26].

Though substantial studies have been performed on BPMN, our effort is distinguished from them in several respects:

(i) The main purpose of our work is to develop a solid foundation for the Petri-like semantics of BPMN rather than on the formal analysis of BPMN models.
(ii) The theoretical basis concentrates on BPMN 1.2 instead of BPMN 1.0. In contrast to BPMN 1.0, BPMN 1.2 supports the concepts of catching events, throwing events and signal events.
(iii) The underlying framework is built on linear temporal logic in lieu of Petri-nets, π-calculus, CSP and NuSMV language.
(iv) The temporal analyses of BPMN process models are a class of problems that are not fully addressed in [7], [8], [10], [14], [15], [16], [17], [18], [20] and [21]. A prime motivation for using linear temporal logic is to complement these studies by providing a methodical approach for reasoning about the temporal aspects of BPMN.
(v) Our semantic analysis examines more advanced graphical elements encompassing subprocess, exception and transaction.

There is relatively little research conducted directly on the execution semantics of BPMN. Dumas et al. [27] introduce a behavioural semantics pertaining to the inclusive merge

V. Lam is with the Computer Centre, The University of Hong Kong, Pokfulam Road, Hong Kong (e-mail: vitus.lam@ieee.org).

(Advance online publication: 27 February 2012)

gateway of BPMN. Börger and Thalheim [28] develop a semantic model for BPMN 1.0 by the abstract state machines method. In [29], Grosskopf presents an executable business process specification language in which the semantics is built on the semantics of BPMN 1.0. Unlike these earlier studies that are based on BPMN 1.0, a full explanation for the behaviour of all the notational elements of BPMN 1.2 is given in Section V. When compared with the work of Dumas et al. that is limited to a single notational element, our work covers a larger set of graphical constructs.

A related strain of research in the context of BPMN deals with the equivalence checking of BPMN processes. In [30], a structured framework for classifying different sorts of equivalences for BPMN processes is presented. As opposed to our previous attempt, the goal of our current work is to construct a mathematical-based semantics that lays the groundwork for analyzing business processes expressed as BPMN.

Besides, there is a great variety of research topics related to BPMN. Auer et al. [31] enhance the BPMN by augmenting it with submit/response-style user interaction. Mazanek and Hanus [32] develop a software tool using the functional logic programming language Curry that automates the transformations between BPMN and BPEL. In [33], Sánchez-González et al. utilize the Bender method to find out the thresholds of the control-flow complexity measures for BPMN process models. Kopp et al. [34] present a systematic way for obtaining a complete BPMN process from a BPMN process fragment. An application of BPMN in the healthcare domain is examined in [35]. Additionally, emerging research on business process management encompasses a broad range of areas such as compliance checking of business constraints [36], automated rectification of behavioural anomalies in business process models [37], process mining [38], [39], [40], [41], declarative workflows [42], [43] and distributed business processes [44].

III. Business Process Modelling Notation

This section is devoted to a discussion of the main features of BPMN 1.2. Most of the material in this section is derived from our earlier studies [30], [45]. For a more thorough description of BPMN, the reader is referred to [1], [2], [3].

The BPMN is a well-known diagrammatic notation for supporting the specification of business processes. The notational elements of BPMN are classified into four types: flow objects, connecting objects, artifacts and swimlanes. The flow objects and connecting objects, which are the basic elements for constructing business processes, are delineated in Figures 1, 2, 3 and 4. The extra information of a business process and the organizational perspective of a business process diagram are expressed in BPMN by means of artifacts (Figure 5) and swimlanes (Figure 6), respectively. Unlike flow objects as well as connecting objects, artifacts and swimlanes are not related to process flows and do not have token-based semantics.

In BPMN, flow objects fall into three categories: events, activities and gateways. The occurrence of a start event (Figure 1) initiates a process and causes the creation of a token that moves down the outgoing sequence flow. Depending on how a none start event is utilized, it represents either (i) the event trigger type is not indicated or (ii) the beginning of a subprocess upon receiving a token from its respective parent process. Contrary to none start events, all other types of start events including message start events, timer start events, conditional start events, signal start events and multiple start events are regarded as trigger-based start events in which the use of them in subprocesses is not permitted.

A message start event, a timer start event, a conditional start event, a signal start event and a multiple start event symbolize the activation of a process on receipt of a message, whenever a particular time-date condition holds, when a condition is fulfilled, upon arrival of a signal and if any one of the two or more defined start events occurs, respectively.

The happening of an end event terminates a process and results in the consumption of a token. A none end event is similar in concept to a none start event. It denotes either (i) the event trigger type is not indicated or (ii) the cessation of a subprocess on receipt of a token along the incoming sequence flow. A message end event, an error end event, a cancel end event, a compensation end event, a signal end event, a terminate end event and a multiple end event signify the completion of a process that leads to the sending of a message to other process, the throwing of an error, the abortion of a transaction, the execution of a compensation, the broadcasting of a signal, the immediate cessation of all process flows and the triggering of all defined end events, respectively.

Catch intermediate events and throw intermediate events are events that occur during the course of a process. With the exception that a catch intermediate event does not initiate a process, the behaviours of a catching none intermediate event, a catching message intermediate event, a catching timer intermediate event, a catching conditional intermediate event, a catching signal intermediate event and a catching multiple intermediate event are analogous to the behaviours of a none start event, a message start event, a timer start event, a conditional start event, a signal start event and a multiple start event. Likewise, the execution semantics of a throw intermediate event is similar to the one of an end event except that a throw intermediate event does not terminate a process. The behavioural semantics of a throwing message intermediate event, a throwing compensation intermediate event, a throwing signal intermediate event and a throwing multiple intermediate event correspond to the behavioural semantics of a message end event, a compensation end event, a signal end event and a multiple end event. The occurrence of an error, the abortion of a transaction, the receipt of a compensation event, the catching of a link event and the throwing of a link event are rendered by a catching error intermediate event, a catching cancel intermediate error, a catching compensation intermediate event, a catching link intermediate event and a throwing link intermediate event.

An activity is regarded as an unit of work that is either a task or a subprocess (Figure 2). A task is atomic and does not support a hierarchical structure. On the contrary, a subprocess is decomposable. The three types of markers in a task are loop, multiple instance and compensation. Likewise, there are five sorts of subprocess markers: collapsed subprocess, loop, multiple instance, ad hoc and compensation. A collapsed subprocess hides all the details of its internal structure, whereas an expanded subprocess shows all the fine details. A transaction is a subprocess in which all the enclosed activities

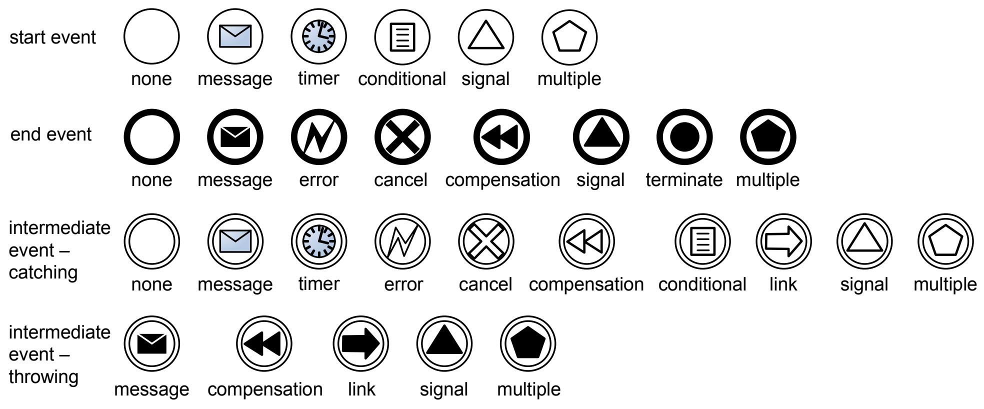

Fig. 1. Events

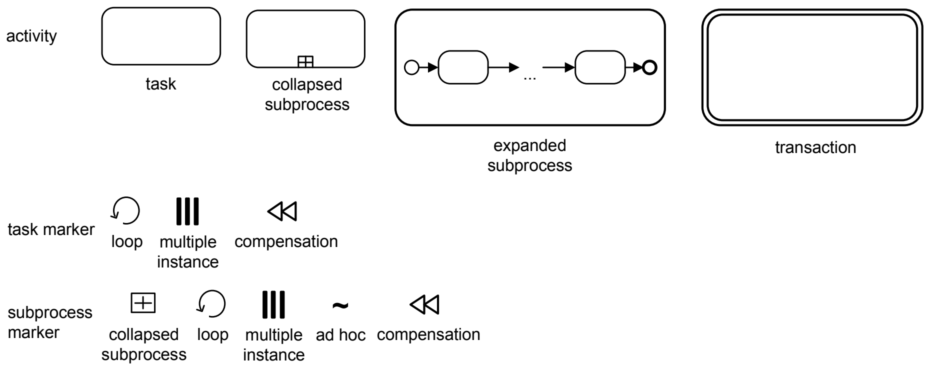

Fig. 2. Activities

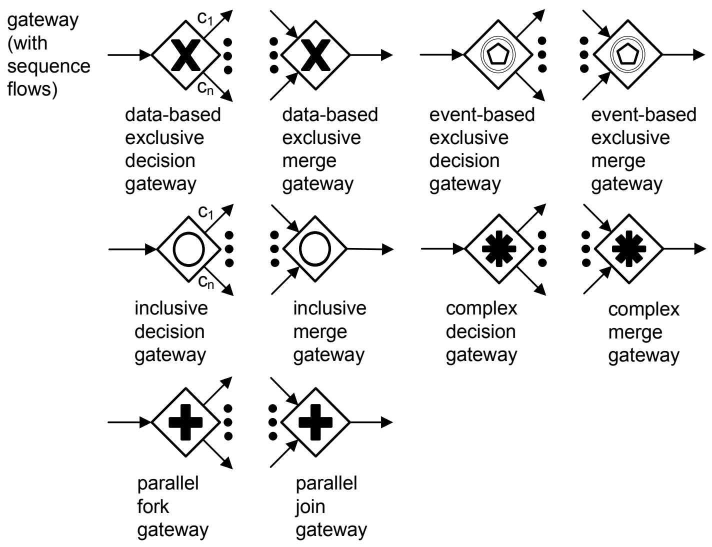
gateway (with sequence flows)
C1
Cn
data-based exclusive decision gateway
data-based exclusive merge gateway
event-based exclusive decision gateway
event-based exclusive merge gateway
inclusive decision gateway
inclusive merge gateway
complex decision gateway
complex merge gateway
parallel fork gateway
parallel join gateway

Fig. 3. Gateways

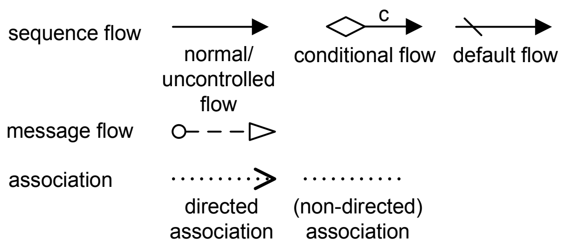

Fig. 4. Connecting objects

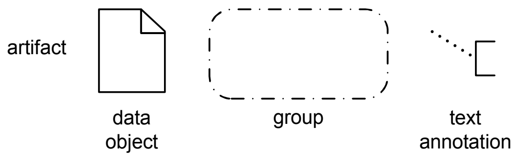

Fig. 5. Artifacts

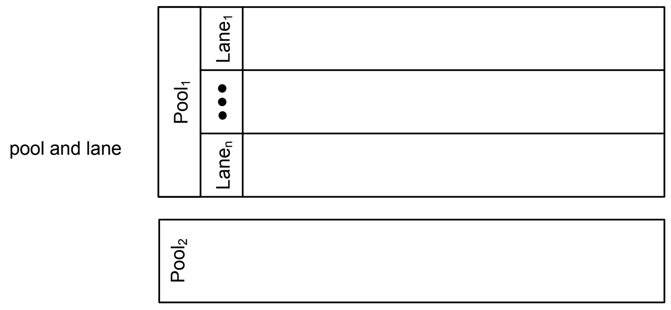

Fig. 6. Swimlanes

are either complete or cancel.

Each outgoing sequence flow of a data-based exclusive decision gateway (Figure 3) is associated with a conditional expression. A token is placed on one of the outgoing sequence flows in which the respective condition is met. On receiving a token from one of the incoming sequence flows, a data-based exclusive merge gateway sends a token down the outgoing sequence flow. When a token reaches an event-based exclusive decision gateway, a token is emitted on each of the outgoing sequence flows. A token leaves an event-based exclusive merge gateway upon receipt of a token along one of the incoming sequence flows.

On receipt of a token, an inclusive decision gateway offers a token to each of the outgoing sequence flows in which the corresponding condition expression returns true. After the arrival of all the tokens generated by an upstream, an inclusive merge gateway sends a token on the outgoing sequence flow. The set of outgoing sequence flows of a complex decision gateway on which tokens are sent as well as the collection of incoming sequence flows of a complex merge gateway from which tokens are received are determined by an expression. The sending of tokens along all outgoing sequence flows by a parallel fork gateway creates parallel flows. The receipt of tokens on all incoming sequence flows by a parallel join gateway synchronizes parallel flows.

Connnecting objects are classified into three categories: sequence flows, message flows and assocations (Figure 4). A sequence flow is a connection between two flow objects in which a token moves from the source flow object to the target flow object. Gateways are passed over in a normal flow. In contrast, an uncontrolled flow does not pass through any gateways. A conditional flow determines whether a token is sent along the outgoing sequence flow by evaluating its associated conditional expression. A default flow is chosen when the conditional expressions of all other outgoing sequence flows become false. A message flow expresses the interaction between two business processes. An association establishes a linkage between a flow object and an artifact. A directed association either (i) specifies data objects that are the inputs and outputs of an activity or (ii) connects a compensation intermediate event attached to the boundary of an activity with a compensation activity. A (non-directional) assocation links up a text annotation and a flow object.

As shown in Figure 5, the three pre-defined artifacts in BPMN are data objects, groups and text annotations. A data object models both electronic and tangible items such as data or document. A group highlights a collection of notational elements for satisfying a variety of purposes encompassing documentation, reporting, analysis, etc. A text annotation offers additional information regarding a process or notational element.

In BPMN, a pool (Figure 6) is considered as a participant. Each pool contains a process and embodies at least one lane. A lane is a means for grouping the different portions of a business process.

IV. Linear Temporal Logic

Temporal logic, which is a category of logic, delineates how the truth value of a formula evolves over time. In the context of computer science, the application of temporal logic is mainly utilized for specifying the desired properties of a system. One of the most commonly accepted ways for classifying temporal logics is by means of time models. In the linear time model there is a single future time at any given point of time, whereas in the branching time model there are multiple future times at any given point of time. The linear temporal logic (LTL) and computation tree logic (CTL) adopt, respectively, the linear time model and branching time model as the underlying structure of time.

By Σ_{AP} we denote the set of atomic propositions over p, q and r. The syntax of LTL is inductively defined as:

$$\phi ::= p \mid \neg\phi \mid (\phi) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \mathrm{X}\phi \mid \mathrm{F}\phi \mid \mathrm{G}\phi \mid \phi\mathrm{U}\phi.$$

X, F, G and U are linear temporal operators. The temporal operators X, F, G and U stand for next state, some future state, all future states and all future states until a condition holds, respectively. The intuitive meanings of the associated LTL formulas are given below:

$\mathrm{X}\phi$: means ϕ is true in next state.

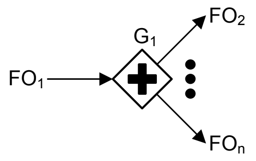

Fig. 7. A parallel gateway G_1

Fϕ:	means ϕ is true in some future state.
Gϕ:	means ϕ is true in all future states.
ϕ_1Uϕ_2:	ϕ_1 is true in all future states until ϕ_2 holds.

The unary operators X, F and G have a higher precedence than the binary operator U.

V. Formal Semantics of BPMN

To facilitate the adoption of LTL as the mathematically founded framework for representing the execution semantics of BPMN, some atomic propositions coupled with their meanings are defined as follows:

$tokenAt_x$:	A token is resided on x.
$tokenAt_{x_1,x_2}$:	A token is located at the sequence flow or directed association connecting x_1 and x_2.
$tokenConsumedBy_{TE}$:	A token is consumed by the terminate end event TE.
$catchEvent_E$:	The event E is triggered.
$throwEvent_E$:	The event E throws a trigger.
c:	The condition c holds.

Our formalized semantics follows the token-based semantics of BPMN given in [2], [3]. Each atomic proposition is utilized to keep track of the status of a token, event or condition.

In BPMN, a gateway is allowed to have multiple incoming and multiple outgoing sequence flows. Since the gateway is regarded as an alternative representation of a merge gateway and a decision gateway in which the outgoing sequence flow of the merge gateway connects to the incoming sequence flow of the decision gateway, we omit it in the semantic mapping.

Figure 7 depicts the graphical representation of a parallel fork gateway G_1. We let F_{PFG} be a set of parallel fork gateways, S_{F} be a set of flow objects, C_{SF} be a set of sequence flows and $F^{\text{Term}}_{\text{EE}}$ be a set of terminate end events. The execution semantics of the parallel fork gateway G_1 connecting to flow objects $FO_1, \ldots, FO_n$ is formally captured in the following definition in terms of an LTL formula on the assumption that there are zero or more terminate end events in the corresponding BPMN processes.

Definition 1 (Parallel Fork Gateway): Suppose $G_1 \in F_{\text{PFG}}$, $FO_i \in S_{\text{F}}$, (FO_1, G_1), $(G_1, FO_j) \in C_{\text{SF}}$ and $TE_k \in F^{\text{Term}}_{\text{EE}}$ for $i = 1, \ldots, n$, $j = 2, \ldots, n$ and $k = 1, \ldots, m$. The execution semantics of the parallel fork gateway G_1 with $n-1$ outgoing sequence flows is denoted in LTL as:

$$\text{G}\Big((tokenAt_{FO_1,G_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \text{X}(\bigwedge_{j=2}^{n} tokenAt_{G_1,FO_j} \wedge \neg tokenAt_{FO_1,G_1})\Big).$$

The atomic proposition $tokenAt_{FO_1,G_1}$ represents the receipt of a token along the incoming sequence flow (FO_1, G_1). The propositional formula $\bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}$ states that none of the terminate end events TE_1, …, TE_m consumes a token which results in the immediate cessation of all flows. The sending of tokens on all outgoing sequence flows is specified by $\bigwedge_{j=2}^{n} tokenAt_{G_1,FO_j}$. The negation of the atomic proposition $tokenAt_{FO_1,G_1}$ stipulates that the received token is no longer on the incoming sequence flow whenever there are tokens on all the outgoing sequence flows. The property that the emission of tokens must take place after receiving a token from the incoming sequence flow is ensured by means of the temporal operator X.

The official semantics of BPMN does not specify whether a maximal set of flow objects is fired in each state of a BPMN diagram. We extend the official semantics by adopting the maximal property as a central principle in our formalization. For instance, if there are tokens on the incoming sequence flows of two parallel fork gateways simultaneously, both gateways are fired in the next state in lieu of either one of them is fired. The temporal operator X in Definition 1 guarantees that the maximal property is preserved.

We denote by F_{PJG} a set of parallel join gateways. The precise semantics of a parallel join gateway is defined below.

Definition 2 (Parallel Join Gateway): Suppose $G_1 \in F_{\text{PJG}}$, $FO_i \in S_{\text{F}}$, (FO_j, G_1), $(G_1, FO_n) \in C_{\text{SF}}$ and $TE_k \in F^{\text{Term}}_{\text{EE}}$ for $i = 1, \ldots, n$, $j = 1, \ldots, n-1$ and $k = 1, \ldots, m$. The behavioural semantics of the parallel join gateway G_1 with $n-1$ incoming sequence flows is expressed as:

$$\text{G}\Big((\bigwedge_{j=1}^{n-1} tokenAt_{FO_j,G_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \text{X}(tokenAt_{G_1,FO_n} \wedge \bigwedge_{j=1}^{n-1} \neg tokenAt_{FO_j,G_1})\Big).$$

The formula $\bigwedge_{j=1}^{n-1} tokenAt_{FO_j,G_1}$ specifies that tokens are received along all incoming sequence flows (FO_j, G_1). The traverse of a token along the outgoing sequence flow (G_1, FO_n) is modelled as the atomic proposition $tokenAt_{G_1,FO_n}$.

Let $F^{\text{D}}_{\text{XDG}}$ be a set of data-based exclusive decision gateways, S_{Cond} be a set of conditions, $\Phi_{\text{Cond}} : C_{\text{SF}} \rightarrow S_{\text{Cond}}$ be a function returning the condition of a sequence flow, $S^{\text{Idx}}_{\text{Cond}} = \{1, \ldots, n-1\}$ and $\Phi_{\text{Eval}} : \{S^{\text{Idx}}_{\text{Cond}}\} \rightarrow S^{\text{Idx}}_{\text{Cond}}$ be a function returning the index of the condition that first evaluates to true.

Definition 3 (Data-based Exclusive Decision Gateway): Suppose $G_1 \in F^{\text{D}}_{\text{XDG}}$, $FO_i \in S_{\text{F}}$, (FO_1, G_1), $(G_1, FO_j) \in C_{\text{SF}}$, $c_{j-1} \in S_{\text{Cond}}$, $\Phi_{\text{Cond}}((G_1, FO_j)) = c_{j-1}$, $t = \Phi_{\text{Eval}}(\{1, \ldots, n-1\})$ and $TE_k \in F^{\text{Term}}_{\text{EE}}$ for $i = 1, \ldots, n$, $j = 2, \ldots, n$ and $k = 1, \ldots, m$. The behaviour of the data-based exclusive decision gateway G_1 with $n-1$ outgoing sequence flows is specified below:

$$\text{G}\Big((tokenAt_{FO_1,G_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k} \wedge c_t) \rightarrow \text{X}(tokenAt_{G_1,FO_{t+1}} \wedge \neg tokenAt_{FO_1,G_1})\Big).$$

The conditional expression of an outgoing sequence flow that is the first to become true is denoted by the atomic proposition c_t. Whenever a token is offered to the incoming sequence flow (FO_1, G_1) and c_t evaluates to true, the data-based exclusive decision gateway G_1 emits a token along the outgoing sequence flow (G_1, FO_{t+1}) represented as the atomic proposition $tokenAt_{G_1,FO_{t+1}}$.

In the following, we present the formal semantics of a data-based exclusive merge gateway. A set of data-based exclusive merge gateways is denoted by $F^{\mathrm{D}}_{\mathrm{XMG}}$.

Definition 4 (Data-based Exclusive Merge Gateway): Suppose $G_1 \in \underline{F^{\mathrm{D}}_{\mathrm{XMG}}}$, $FO_i \in S_{\mathrm{F}}$, (FO_j, G_1), $(G_1, FO_n) \in C_{\mathrm{SF}}$ and $TE_k \in F^{\mathrm{Term}}_{\mathrm{EE}}$ for $i = 1,\ldots,n$, $j = 1,\ldots,n-1$ and $k = 1,\ldots,m$. The execution semantics of the data-based exclusive merge gateway G_1 is defined by:

$$\mathrm{G}\Bigg(\bigwedge_{j=1}^{n-1}\Big((tokenAt_{FO_j,G_1} \wedge \bigwedge_{k=1}^{m}\neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{G_1,FO_n} \wedge \neg tokenAt_{FO_j,G_1})\Big)\Bigg).$$

The behaviour of the data-based exclusive merge gateway G_1 is exclusive in the sense that a token is sent along the outgoing sequence flow as soon as a token is received from one of the incoming sequence flows. The formula $tokenAt_{FO_j,G_1} \wedge \bigwedge_{k=1}^{m}\neg tokenConsumedBy_{TE_k}$ formalizes the exclusive behaviour of the gateway since it is based on one incoming sequence flow rather than all incoming sequence flows.

Next, the mathematical definitions of event-based exclusive decision gateway and event-based exclusive merge gateway are given. The sets of event-based exclusive decision gateways and event-based exclusive merge gateways are denoted, respectively, by $F^{\mathrm{E}}_{\mathrm{XDG}}$ and $F^{\mathrm{E}}_{\mathrm{XMG}}$. We define Γ_{IE} = {None, Msg, Timer, Err, Cncl, Cmpen, Cond, Link, Sign, Multi}, $F^{\mathrm{None}}_{\mathrm{IE}}$ as a set of none intermediate events for catching the event triggers, $F^{\mathrm{Msg}}_{\mathrm{IE}}$ as a set of message intermediate events for catching the event triggers, $F^{\mathrm{Timer}}_{\mathrm{IE}}$ as a set of timer intermediate events for catching the event triggers, $F^{\mathrm{Err}}_{\mathrm{IE}}$ as a set of error intermediate events for catching the event triggers, $F^{\mathrm{Cncl}}_{\mathrm{IE}}$ as a set of cancel intermediate events for catching the event triggers, $F^{\mathrm{Cmpen}}_{\mathrm{IE}}$ as a set of compensation intermediate events for catching the event triggers, $F^{\mathrm{Cond}}_{\mathrm{IE}}$ as a set of conditional intermediate events for catching the event triggers, $F^{\mathrm{Link}}_{\mathrm{IE}}$ as a set of link intermediate events for catching the event triggers, $F^{\mathrm{Sign}}_{\mathrm{IE}}$ as a set of signal intermediate events for catching the event triggers and $F^{\mathrm{Multi}}_{\mathrm{IE}}$ as a set of multiple intermediate events for catching the event triggers.

Definition 5 (Event-based Exclusive Decision Gateway): Suppose $G_1 \in F^{\mathrm{E}}_{\mathrm{XDG}}$, $FO_i \in S_{\mathrm{F}}$, $E_j \in \bigcup_{\underline{\sigma\in\Gamma_{\mathrm{IE}}}} F^{\sigma}_{\mathrm{IE}}$, (FO_1, G_1), (G_1, E_j), $(E_j, FO_{j+1}) \in C_{\mathrm{SF}}$ and $TE_k \in F^{\mathrm{Term}}_{\mathrm{EE}}$ for $i = 1,\ldots,n$, $j = 1,\ldots,n-1$ and $k = 1,\ldots,m$. The LTL representation of the event-based exclusive decision gateway G_1 with $n-1$ outgoing sequence flows is given below:

$$\mathrm{G}\Bigg(\Big((tokenAt_{FO_1,G_1} \wedge \bigwedge_{k=1}^{m}\neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(\bigwedge_{j=1}^{n-1} tokenAt_{G_1,E_j} \wedge \neg tokenAt_{FO_1,G_1})\Big)\wedge \Big(\bigwedge_{j=1}^{n-1}\Big((tokenAt_{G_1,E_j} \wedge \bigwedge_{k=1}^{m}\neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{E_j} \wedge \neg tokenAt_{G_1,E_j})\Big)\Big)\wedge \Big(\bigwedge_{j=1}^{n-1}\Big((tokenAt_{E_j} \wedge catchEvent_{E_j} \wedge \bigwedge_{i\in\{1,\ldots,n-1\}\backslash\{j\}}\neg catchEvent_{E_i} \wedge \bigwedge_{k=1}^{m}\neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{E_j,FO_{j+1}} \wedge \bigwedge_{j=1}^{n-1}\neg tokenAt_{E_j})\Big)\Big)\Bigg).$$

On receiving a token along the incoming sequence flow, a token traverses each of the outgoing sequence flows of the event-based exclusive decision gateway G_1 expressed as $\bigwedge_{j=1}^{n-1} tokenAt_{G_1,E_j}$. The atomic proposition $tokenAt_{E_j}$ signifies the arrival of the token at the catch intermediate $event_{E_j}$. When one of the catch intermediate events E_j is triggered as defined in LTL by $tokenAt_{E_j} \wedge catchEvent_{E_j} \wedge \bigwedge_{i\in\{1,\ldots,n-1\}\backslash\{j\}}\neg catchEvent_{E_i}$, the token on E_j moves down the outgoing sequence flow (E_j, FO_{j+1}) denoted as $tokenAt_{E_j,FO_{j+1}}$. The formula $\bigwedge_{j=1}^{n-1}\neg tokenAt_{E_j}$ symbolizes the token exits the catch intermediate event E_j and the consumption of all tokens on the other catch intermediate events.

Definition 6 (Event-based Exclusive Merge Gateway): Suppose $G_1 \in \underline{F^{\mathrm{E}}_{\mathrm{XMG}}}$, $FO_i \in S_{\mathrm{F}}$, (FO_j, G_1), $(G_1, FO_n) \in C_{\mathrm{SF}}$ and $TE_k \in F^{\mathrm{Term}}_{\mathrm{EE}}$ for $i = 1,\ldots,n$, $j = 1,\ldots,n-1$ and $k = 1,\ldots,m$. The event-based exclusive merge gateway G_1 with $n-1$ incoming sequence flows is encoded in the same way as a data-based exclusive merge gateway.

An event-based exclusive merge gateway and a data-based exclusive merge gateway are equivalent in terms of behavioural semantics. As a result of this, the encodings of these two gateways are the same as stipulated by Definitions 6 and 4.

The definition that follows comprises two parts. The first part concentrates on an inclusive decision gateway without a default flow. The second part is concerned with an inclusive decision gateway that a default sequence flow is specified. We let F_{IDG} be a set of inclusive decision gateways and $\Phi_{\mathrm{IsDf}} : C_{\mathrm{SF}} \rightarrow \mathbb{B}$ be a function that returns whether a sequence flow is a default flow.

Definition 7 (Inclusive Decision Gateway): Suppose $G_1 \in F_{\mathrm{IDG}}$, $FO_i \in S_{\mathrm{F}}$, (FO_1, G_1), $(G_1, FO_j) \in C_{\mathrm{SF}}$, $S_{\mathrm{Df}^{(G_1)}} = \{x | x \in \bigcup_{i\in\{2,\ldots,n\}}\{(G_1, FO_i)\} \wedge \Phi_{\mathrm{IsDf}}(x) = true\}$ and $TE_k \in \underline{F^{\mathrm{Term}}_{\mathrm{EE}}}$ for $i = 1, \ldots, n$, $j = 2,\ldots,n$ and $k = 1, \ldots, m$. If $c_{j-1} \in S_{\mathrm{Cond}}$, $C^{\mathrm{All}}_{\mathrm{T}} = 2^{\{1,\ldots,n-1\}} \setminus \{\emptyset\}$, $C_{\mathrm{T}} \in C^{\mathrm{All}}_{\mathrm{T}}$, $\Phi_{\mathrm{Cond}}((G_1, FO_j)) = c_{j-1}$ and $S_{\mathrm{Df}^{(G_1)}} = \emptyset$ for $j = 2, \ldots, n$, then the inclusive decision gateway G_1 with no default flow and $n-1$ outgoing sequence flows associated with conditions c_{j-1} is modelled in LTL as:

$$\mathrm{G}\Big((tokenAt_{FO_1,G_1} \wedge \bigwedge_{k=1}^{m}\neg tokenConsumedBy_{TE_k} \wedge \bigwedge_{i\in C_{\mathrm{T}}} c_i \wedge \bigwedge_{i\in\{1,\ldots,n-1\}\backslash C_{\mathrm{T}}}\neg c_i) \rightarrow \mathrm{X}(\bigwedge_{i\in C_{\mathrm{T}}} tokenAt_{G_1,FO_{i+1}} \wedge \neg tokenAt_{FO_1,G_1})\Big).$$

If $c_{j-1} = S_{\mathrm{Cond}}$, $C^{\mathrm{All}}_{\mathrm{T}} = 2^{\{1,\ldots,n-2\}}$, $C_{\mathrm{T}} \in C^{\mathrm{All}}_{\mathrm{T}}$, $\Phi_{\mathrm{Cond}}((G_1, FO_j))$

$= c_{j-1}$ and $S_{\mathrm{Df}^{(G_1)}} = \{(G_1, FO_n)\}$ for $j = 2, \ldots, n-1$, then the inclusive decision gateway G_1 associated with conditions c_{j-1} and a default flow (G_1, FO_n) is defined in LTL by:

$$\mathrm{G}\Bigg(\bigg((tokenAt_{FO_1,G_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k} \wedge \bigwedge_{i \in C_{\mathrm{T}}} c_i \wedge \bigwedge_{i \in \{1,\ldots,n-2\} \setminus C_{\mathrm{T}}} \neg c_i \wedge \neg(C_{\mathrm{T}} = \emptyset)) \rightarrow \mathrm{X}(\bigwedge_{i \in C_{\mathrm{T}}} tokenAt_{G_1,FO_{i+1}} \wedge \neg tokenAt_{FO_1,G_1})\bigg) \wedge \bigg((tokenAt_{FO_1,G_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k} \wedge \bigwedge_{i \in C_{\mathrm{T}}} c_i \wedge \bigwedge_{i \in \{1,\ldots,n-2\} \setminus C_{\mathrm{T}}} \neg c_i \wedge (C_{\mathrm{T}} = \emptyset)) \rightarrow \mathrm{X}(tokenAt_{G_1,FO_n} \wedge \neg tokenAt_{FO_1,G_1})\bigg)\Bigg).$$

The default sequence flow of the gateway G_1 is denoted by $S_{\mathrm{Df}^{(G_1)}}$. In the first portion of the definition, $S_{\mathrm{Df}^{(G_1)}}$ is an empty set as there is no default flow. In contrast, the set $S_{\mathrm{Df}^{(G_1)}}$ in the second portion of the definition contains (G_1, FO_n) in order to indicate that it is a default flow.

The set $C_{\mathrm{T}}^{\mathrm{All}}$ in the above represents all possible combinations of conditions that hold. To conform with the semantics that at least one of the conditions must hold, the empty set is excluded from the set $C_{\mathrm{T}}^{\mathrm{All}}$ when there is no default flow. The set C_{T}, which is an element of $C_{\mathrm{T}}^{\mathrm{All}}$, is the collection of conditions that are evaluated to true.

The use of the set C_{T} in both formulas $\bigwedge_{i \in C_{\mathrm{T}}} c_i$ and $\bigwedge_{i \in C_{\mathrm{T}}} tokenAt_{G_1,FO_{i+1}}$ guarantees that every activation of an outgoing sequence flow is due to the condition associated with the outgoing sequence flow is evaluated to true. A token moves down the default sequence flow represented as $tokenAt_{G_1,FO_n}$ in the second temporal formula if $C_{\mathrm{T}} = \emptyset$ is true.

The BPMN specification does not stipulate how the activation of an inclusive merge gateway is decided. Dumas et al. [27] close the gap by proposing a method that finds out whether an inclusive merge gateway is enabled in linear time. In what follows, we build upon their work and abstract away the detailed implementation of the propounded algorithm. We define F_{IMG} as a set of inclusive merge gateways and $I_{\mathrm{TKN}} \in 2^{\{1,\ldots,n-1\}} \setminus \{\emptyset\}$ as the set of incoming sequence flows that tokens are expected to be received from an upstream whenever an inclusive merge gateway is activated. The set I_{TKN} is obtained by checking which incoming sequence flows have one or more tokens.

Definition 8 (Inclusive Merge Gateway): Suppose $G_1 \in F_{\mathrm{IMG}}$, $FO_i \in S_{\mathrm{F}}$, $(FO_j, G_1), (G_1, FO_n) \in C_{\mathrm{SF}}$ and $TE_k \in \overline{F_{\mathrm{EE}}^{\mathrm{Term}}}$ for $i = 1, \ldots, n$, $j = 1, \ldots, n-1$ and $k = 1, \ldots, m$. The encoding of the behaviour of the inclusive merge gateway G_1 is given by:

$$\mathrm{G}\Big((\bigwedge_{j \in I_{\mathrm{TKN}}} tokenAt_{FO_j,G_1} \wedge \bigwedge_{i \in \{1,\ldots,n-1\} \setminus I_{\mathrm{TKN}}} \neg tokenAt_{FO_i,G_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{G_1,FO_n} \wedge \bigwedge_{j \in I_{\mathrm{TKN}}} \neg tokenAt_{FO_j,G_1})\Big).$$

The receipt of one token on each incoming sequence flow that a token is expected to arrive is detected by $\bigwedge_{j \in I_{\mathrm{TKN}}} tokenAt_{FO_j,G_1}$. The emission of a token along the outgoing flow (G_1, FO_n) after synchronizing the set of incoming sequence flows I_{TKN} is represented as $tokenAt_{G_1,FO_n}$.

In [46], Völzer puts forwards a brand-new algorithm with a linear-time complexity for determining if an inclusive merge gateway is activated. The use of this new algorithm in substitution for the one advocated by Dumas et al. is valid as concrete implementation is not specified in Definition 8.

Let F_{CDG} be a set of complex decision gateways and $O_{\mathrm{Actd}} \in 2^{\{2,\ldots,n\}} \setminus \{\emptyset\}$ be the set of outgoing sequence flows that are selected by evaluating the expression of a complex decision gateway.

Definition 9 (Complex Decision Gateway): Suppose $G_1 \in F_{\mathrm{CDG}}$, $FO_i \in S_{\mathrm{F}}$, (FO_1, G_1), $(G_1, FO_j) \in C_{\mathrm{SF}}$ and $TE_k \in \overline{F_{\mathrm{EE}}^{\mathrm{Term}}}$ for $i = 1, \ldots, n$, $j = 2, \ldots, n$ and $k = 1, \ldots, m$. The behaviour of the complex decision gateway G_1 with $n-1$ outgoing sequence flows is represented as:

$$\mathrm{G}\Big((tokenAt_{FO_1,G_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(\bigwedge_{j \in O_{\mathrm{Actd}}} tokenAt_{G_1,FO_j} \wedge \neg tokenAt_{FO_1,G_1})\Big).$$

An inclusive decision gateway and a complex decision gateway both activate one of the combinations of the outgoing sequence flows. Unlike an inclusive decision gateway, a complex decision gateway determines the collection of outgoing sequence flows to be selected by evaluating an expression. The result values of the evaluation are contained in the set O_{Actd}. As O_{Actd} is a non-empty set, the behavioural semantics that one or more outgoing sequence flows are activated by a complex decision gateway is preserved. The activation of the outgoing sequence flows is modelled as $\bigwedge_{j \in O_{\mathrm{Actd}}} token_{G_1,FO_j}$ in Definition 9.

Compared to a data-based exclusive merge gateway, an event-based exclusive merge gateway and an inclusive merge gateway, the exact behavioural semantics of a complex merge gateway depends on an expression in lieu of a predefined merging behaviour. In the following definition, an upstream parallel fork gateway is utilized to illustrate the use of a complex merge gateway for implementing a discriminator pattern. Our attention is confined to this case as other merging patterns of a complex merge gateway are merely variants of the formalized execution semantics. By F_{CMG} we denote a set of complex merge gateways.

Definition 10 (Complex Merge Gateway): Suppose $G_1 \in F_{\mathrm{CMG}}$, $FO_i \in S_{\mathrm{F}}$, (FO_j, G_1), $(G_1, FO_n) \in C_{\mathrm{SF}}$ and $TE_k \in \overline{F_{\mathrm{EE}}^{\mathrm{Term}}}$ for $i = 1, \ldots, n$, $j = 1, \ldots, n-1$ and $k = 1, \ldots, m$. If an upstream parallel fork gateway is placed prior to the complex merge gateway G_1 for modelling a discriminator pattern, then the execution semantics of the complex merge gateway G_1 is specified as:

$$\mathrm{G}\Bigg(\bigwedge_{j_1=1}^{n-1}\Bigg(\big((tokenAt_{FO_{j_1},G_1} \wedge \bigwedge_{i_1\in\{1,\ldots,n-1\}\setminus\{j_1\}} \neg tokenAt_{FO_{i_1},G_1} \wedge$$
$$\bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow$$
$$\mathrm{X}\Big(tokenAt_{G_1,FO_n} \wedge \neg tokenAt_{FO_{j_1},G_1} \wedge$$
$$\bigwedge_{j_2\in\{1,\ldots,n-1\}\setminus\{j_1\}} ((tokenAt_{FO_{j_2},G_1} \wedge$$
$$\bigwedge_{i_2\in\{1,\ldots,n-1\}\setminus\{j_2\}} \neg tokenAt_{FO_{i_2},G_1} \wedge$$
$$\bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow$$
$$\mathrm{X}(\neg tokenAt_{FO_{j_2},G_1})))\Big)\Bigg)\Bigg).$$

The most important feature of a discriminator pattern is to traverse the first received token along the outgoing sequence flow and discard all other received tokens. The former is expressed as $tokenAt_{G_1,FO_n}$ and the latter is formalized by:

$$\bigwedge_{j_2\in\{1,\ldots,n-1\}\setminus\{j_1\}} \Big((tokenAt_{FO_{j_2},G_1} \wedge$$
$$\bigwedge_{i_2\in\{1,\ldots,n-1\}\setminus\{j_2\}} \neg tokenAt_{FO_{i_2},G_1} \wedge$$
$$\bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow$$
$$\mathrm{X}(\neg tokenAt_{FO_{j_2},G_1})\Big).$$

In the same spirit, the execution semantics of start event, catch intermediate event, throw intermediate event and end event is further specified in the form of LTL formulas. We define Γ_{SE} = {None, Msg, Timer, Cond, Sign, Multi}. Let $F_{\mathrm{SE}}^{\mathrm{None}}$ be a set of none start events, $F_{\mathrm{SE}}^{\mathrm{Msg}}$ be a set of message start events, $F_{\mathrm{SE}}^{\mathrm{Timer}}$ be a set of timer start events, $F_{\mathrm{SE}}^{\mathrm{Cond}}$ be a set of conditional start events, $F_{\mathrm{SE}}^{\mathrm{Sign}}$ be a set of signal start events and $F_{\mathrm{SE}}^{\mathrm{Multi}}$ be a set of multiple start events.

Definition 11 (Start Event): Suppose $E_1 \in \bigcup_{\sigma\in\Gamma_{\mathrm{SE}}} F_{\mathrm{SE}}^{\sigma}$, $FO_1 \in S_{\mathrm{F}}$ and $(E_1, FO_1) \in C_{\mathrm{SF}}$. The behavioural semantics of the start event E_1 is encoded as:

$$\mathrm{G}\Big((catchEvent_{E_1} \rightarrow \mathrm{X}tokenAt_{E_1}) \wedge$$
$$\Big(tokenAt_{E_1} \rightarrow \mathrm{X}(tokenAt_{E_1,FO_1} \wedge \neg tokenAt_{E_1})\Big)\Big).$$

A token is created whenever a start event trigger occurs. The LTL formula $catchEvent_{E_1} \rightarrow \mathrm{X}tokenAt_{E_1}$ captures the relationship between the occurrence of a start event trigger and the generation of a token. The generated token then leaves the start event and traverses the outgoing sequence flow. These correspond to the formula $tokenAt_{E_1} \rightarrow \mathrm{X}(tokenAt_{E_1,FO_1} \wedge \neg tokenAt_{E_1})$.

Definition 12 (Catch Intermediate Event): Suppose $E_1 \in \bigcup_{\sigma\in\Gamma_{\mathrm{IE}}\setminus\{\mathrm{Cmpen}\}} F_{\mathrm{IE}}^{\sigma}$, FO_1, $FO_2 \in S_{\mathrm{F}}$, (FO_1, E_1), $(E_1, FO_2) \in C_{\mathrm{SF}}$ and $TE_k \in F_{\mathrm{EE}}^{\mathrm{Term}}$ for $k = 1, \ldots, m$. The behaviour of the catch intermediate event E_1 is modelled by:

$$\mathrm{G}\Bigg(\Big((tokenAt_{FO_1,E_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow$$
$$\mathrm{X}(tokenAt_{E_1} \wedge \neg tokenAt_{FO_1,E_1})\Big) \wedge$$
$$\Big((tokenAt_{E_1} \wedge catchEvent_{E_1} \wedge$$
$$\bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow$$
$$\mathrm{X}(tokenAt_{E_1,FO_2} \wedge \neg tokenAt_{E_1})\Big)\Bigg).$$

A catch compensation intermediate event is not allowed to utilize in a normal flow. This is expressed as $\sigma \in \Gamma_{\mathrm{SE}} \setminus$ {Cmpen}. The arrival of a token at E_1, the placement of the token on E_1 and the occurrence of the intermediate event trigger are denoted by $tokenAt_{FO_1,E_1}$, $tokenAt_{E_1}$ and $catchEvent_{E_1}$, respectively.

We define $\Gamma_{\overline{\mathrm{IE}}} = \{\overline{\mathrm{Msg}}, \overline{\mathrm{Cmpen}}, \overline{\mathrm{Link}}, \overline{\mathrm{Sign}}, \overline{\mathrm{Multi}}\}$, $F_{\mathrm{IE}}^{\overline{\mathrm{Msg}}}$ to be a set of message intermediate events for throwing the event triggers, $F_{\mathrm{IE}}^{\overline{\mathrm{Cmpen}}}$ to be a set of compensation intermediate events for throwing the event triggers, $F_{\mathrm{IE}}^{\overline{\mathrm{Link}}}$ to be a set of link intermediate events for throwing the event triggers, $F_{\mathrm{IE}}^{\overline{\mathrm{Sign}}}$ to be a set of signal intermediate events for throwing the event triggers, $F_{\mathrm{IE}}^{\overline{\mathrm{Multi}}}$ to be a set of multiple intermediate events for throwing the event triggers.

Definition 13 (Throw Intermediate Event): Suppose $E_1 \in \bigcup_{\sigma\in\Gamma_{\overline{\mathrm{IE}}}} F_{\mathrm{IE}}^{\sigma}$, FO_1, $FO_2 \in S_{\mathrm{F}}$, (FO_1, E_1), $(E_1, FO_2) \in C_{\mathrm{SF}}$ and $TE_k \in F_{\mathrm{EE}}^{\mathrm{Term}}$ for $k = 1, \ldots, m$. The throw intermediate event E_1 is specified in LTL as:

$$\mathrm{G}\Bigg(\Big((tokenAt_{FO_1,E_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow$$
$$\mathrm{X}(tokenAt_{E_1} \wedge \neg tokenAt_{FO_1,E_1})\Big) \wedge$$
$$\Big((tokenAt_{E_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow$$
$$\mathrm{X}(throwEvent_{E_1} \wedge tokenAt_{E_1,FO_2} \wedge \neg tokenAt_{E_1})\Big)\Bigg).$$

In contrast to a catch intermediate event, a throw intermediate event is triggered immediately on receipt of a token. The firing of the intermediate event E_1 is encoded in LTL as $throwEvent_{E_1}$.

Let Γ_{EE} = {None, Msg, Err, Cncl, Cmpen, Sign, Term, Multi}. Assume a set of none end events $F_{\mathrm{EE}}^{\mathrm{None}}$, a set of message end events $F_{\mathrm{EE}}^{\mathrm{Msg}}$, a set of error end events $F_{\mathrm{EE}}^{\mathrm{Err}}$, a set of cancel end events $F_{\mathrm{EE}}^{\mathrm{Cncl}}$, a set of compensation end events $F_{\mathrm{EE}}^{\mathrm{Cmpen}}$, a set of signal end events $F_{\mathrm{EE}}^{\mathrm{Sign}}$, a set of terminate end events $F_{\mathrm{EE}}^{\mathrm{Term}}$ and a set of multiple end events $F_{\mathrm{EE}}^{\mathrm{Multi}}$.

Definition 14 (End Event): Suppose $E_1 \in \bigcup_{\sigma\in\Gamma_{\mathrm{EE}}} F_{\mathrm{EE}}^{\sigma}$, $FO_1 \in S_{\mathrm{F}}$, $(FO_1, E_1) \in C_{\mathrm{SF}}$ and $TE_k \in F_{\mathrm{EE}}^{\mathrm{Term}}$ for $k = 1, \ldots, m$. If $E_1 \in F_{\mathrm{EE}}^{\mathrm{None}}$, then the execution semantics of the end event E_1 is represented in LTL as:

$$\mathrm{G}\Big(\Big((tokenAt_{FO_1,E_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{E_1} \wedge \neg tokenAt_{FO_1,E_1})\Big) \wedge \Big(tokenAt_{E_1} \rightarrow \mathrm{X}(tokenConsumedBy_{E_1} \wedge \neg tokenAt_{E_1})\Big)\Big).$$

If $E_1 \in \bigcup_{\sigma\in\Gamma_{\mathrm{EE}}\setminus\{\mathrm{None}\}} F_{\mathrm{EE}}^{\overline{\sigma}}$, then the end event E_1 is expressed as:

$$\mathrm{G}\Big(\Big((tokenAt_{FO_1,E_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{E_1} \wedge \neg tokenAt_{FO_1,E_1})\Big) \wedge \Big(tokenAt_{E_1} \rightarrow \mathrm{X}(throwEvent_{E_1} \wedge tokenConsumedBy_{E_1} \wedge \neg tokenAt_{E_1})\Big)\Big).$$

Upon receipt of a token from the incoming sequence flow (FO_1, E_1), the end event E_1 consumes the token as stated in both the first and second LTL formulas. As opposed to a none end event that does not result in the throwing of an event trigger, end event results which occur in other types of end events are modelled as $throwEvent_{E_1}$ in the second LTL formula.

We leave the topic of the formal definitions of events for now and move on to tasks and subprocesses. Assume a set of tasks is denoted by F_{T}.

Definition 15 (Task): Suppose $A_1 \in F_{\mathrm{T}}$, $FO_i \in S_{\mathrm{F}}$ and $TE_k \in F_{\mathrm{EE}}^{\overline{\mathrm{Term}}}$ for $i = 1, \ldots, n$ and $k = 1, \ldots, m$. If (FO_j, A_1), $(A_1, FO_n) \in C_{\mathrm{SF}}$ for $j = 1, \ldots, n-1$, then the LTL encoding of the task A_1 with $n-1$ incoming sequence flows is shown below:

$$\mathrm{G}\Big(\Big((tokenAt_{FO_1,A_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{A_1} \wedge \neg tokenAt_{FO_1,A_1})\Big) \wedge \vdots \Big((tokenAt_{FO_{n-1},A_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{A_1} \wedge \neg tokenAt_{FO_{n-1},A_1})\Big) \wedge \Big((tokenAt_{A_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{F}(tokenAt_{A_1,FO_n})\Big)\Big).$$

If (FO_1, A_1), $(A_1, FO_j) \in C_{\mathrm{SF}}$ for $j = 2, \ldots, n$, then the behaviour of the task A_1 with $n-1$ outgoing sequence flows is expressed in LTL as:

$$\mathrm{G}\Big(\Big((tokenAt_{FO_1,A_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{A_1} \wedge \neg tokenAt_{FO_1,A_1})\Big) \wedge$$

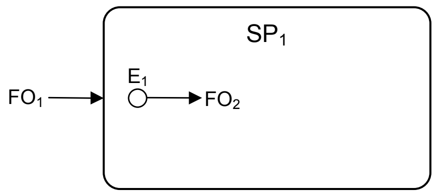

Fig. 8. The initiation of a subprocess SP_1

$$\Big((tokenAt_{A_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{F}(\bigwedge_{i=2}^{n} tokenAt_{A_1,FO_i} \wedge \neg tokenAt_{A_1})\Big)\Big).$$

Consider the task A_1 with multiple incoming sequence flows and one outgoing sequence flow. The formal behavioural semantics is encoded as the first formula. The execution of the task A_1 commences whenever a token is received from one of the incoming sequence flows (FO_j, A_1). The associated LTL representation is defined by the following formulas:

$$(tokenAt_{FO_1,A_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{A_1} \wedge \neg tokenAt_{FO_1,A_1})$$
$$\vdots$$
$$(tokenAt_{FO_{n-1},A_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{A_1} \wedge \neg tokenAt_{FO_{n-1},A_1}).$$

A token is emitted on the outgoing sequence flow after the termination of the task A_1. Since the duration for performing a task varies, a temporal operator F is utilized to indicate that the atomic proposition $tokenAt_{A_1,FO_n}$ holds solely at some point in the future.

The LTL formula of the task A_1 with one incoming sequence flow and multiple outgoing sequence flows can be obtained by combining the preceding formula:

$$(tokenAt_{FO_1,A_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow \mathrm{X}(tokenAt_{A_1} \wedge \neg tokenAt_{FO_1,A_1})$$

with the definition of a parallel fork gateway as well as substituting the temporal operator F for X in the formula of Definition 1. This is due to the facts that (i) a task with one incoming sequence flow is just a special case of a task with multiple incoming sequence flows; and (ii) the traverse of a token along each outgoing sequence flow of a task is similar to the behaviour of a parallel fork gateway.

In BPMN, the notion of hierarchical structures is underpinned by subprocesses. Figures 8 and 9 delineate, respectively, the initiation and termination of a subprocess.

When a token arrives from the flow object FO_1 (Figure 8), the parent-level token is placed inside the subprocess SP_1 and the none start event E_1 is triggered. A child-level token is then generated and offered to the outgoing sequence flow that connect E_1 and the flow object FO_2.

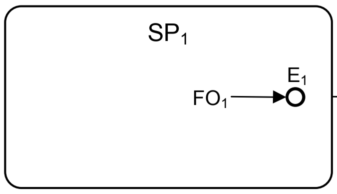

Fig. 9. The termination of a subprocess SP_1

In Figure 9, the receipt of the child-level token along the incoming sequence flow emanating from the flow object FO_1 triggers the none end events E_1. The none end event E_1 consumes the child-level token. The subprocess SP_1 terminates when all other child-level tokens are consumed and the parent-level token is then sent down the outgoing sequence flow connecting SP_1 and the flow object FO_2.

Let $F_{\mathrm{SP}}^{\mathrm{Embed}}$ be a set of embedded subprocesses, $F_{\mathrm{SP}}^{\mathrm{Reuse}}$ be a set of reusable subprocesses, $F_{\mathrm{SP}}^{\mathrm{Ref}}$ be a set of reference subprocesses and Φ_{IsTX}: $F_{\mathrm{SP}}^{\mathrm{Embed}} \cup F_{\mathrm{SP}}^{\mathrm{Reuse}} \cup F_{\mathrm{SP}}^{\mathrm{Ref}} \rightarrow \mathbb{B}$ be a function that returns whether a subprocess is a transaction or not. We denote by $S_{\mathrm{F}^{(SP_1)}}$, $F_{\mathrm{SE}^{(SP_1)}}^{\mathrm{None}}$, $F_{\mathrm{EE}^{(SP_1)}}^{\mathrm{None}}$, $F_{\mathrm{EE}^{(SP_1)}}^{\mathrm{Term}}$ and $C_{\mathrm{SF}^{(SP_1)}}$ the sets of flow objects, none start events, none end events, terminate end events and sequence flows of SP_1.

Definition 16 (Subprocess): Suppose $SP_1 \in F_{\mathrm{SP}}^{\mathrm{Embed}} \cup F_{\mathrm{SP}}^{\mathrm{Reuse}} \cup F_{\mathrm{SP}}^{\mathrm{Ref}}$, $\Phi_{\mathrm{IsTX}}(SP_1) = \mathrm{false}$, $FO_1 \in S_{\mathrm{F}}$, $FO_2 \in S_{\mathrm{F}^{(SP_1)}}$, $TE_k \in F_{\mathrm{EE}}^{\mathrm{Term}}$ for $k = 1, \ldots, m$. If $E_1 \in F_{\mathrm{SE}^{(SP_1)}}^{\mathrm{None}}$, $(FO_1, SP_1) \in C_{\mathrm{SF}}$ and $(E_1, FO_2) \in C_{\mathrm{SF}^{(SP_1)}}$, then the initiation of the subprocess SP_1 is formally defined as follows:

$$\begin{aligned}
\mathrm{G}\Bigg(&\Big(\big(tokenAt_{FO_1,SP_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}\big) \rightarrow \\
&\quad \mathrm{X}(tokenAt_{SP_1} \wedge throwEvent_{E_1} \wedge \neg tokenAt_{FO_1,SP_1})\Big) \wedge \\
&\Big(catchEvent_{E_1} \rightarrow \mathrm{X}(tokenAt_{E_1})\Big) \wedge \\
&\Big(tokenAt_{E_1} \rightarrow \mathrm{X}(tokenAt_{E_1,FO_2} \wedge \neg tokenAt_{E_1})\Big)\Bigg).
\end{aligned}$$

If $E_i \in F_{\mathrm{EE}^{(SP_1)}}^{\mathrm{None}}$, $(FO_1, E_1) \in C_{\mathrm{SF}^{(SP_1)}}$, $(SP_1, FO_2) \in C_{\mathrm{SF}}$ and $TE_j^{SP_1} \in F_{\mathrm{EE}^{(SP_1)}}^{\mathrm{Term}}$ for $i = 1, \ldots, n_1$ and $j = 1, \ldots, n_2$, then the behaviour of the termination of the subprocess SP_1 is denoted as:

$$\begin{aligned}
\mathrm{G}\Bigg(&\Big(\big(tokenAt_{FO_1,E_1} \wedge \bigwedge_{j=1}^{n_2} \neg tokenConsumedBy_{TE_j^{SP_1}} \wedge \\
&\quad \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}\big) \rightarrow \\
&\quad \mathrm{X}(tokenAt_{E_1} \wedge \neg tokenAt_{FO_1,E_1})\Big) \wedge \\
&\Big(tokenAt_{E_1} \rightarrow \mathrm{X}(tokenConsumedBy_{TE_1} \wedge \neg tokenAt_{E_1})\Big) \wedge \\
&\Big(\big(\bigwedge_{i=1}^{n_1} tokenConsumedBy_{E_i} \wedge tokenAt_{SP_1}\big) \rightarrow \\
&\quad \mathrm{X}(\neg tokenAt_{SP_1} \wedge tokenAt_{SP_1,FO_2})\Big)\Bigg).
\end{aligned}$$

The placement of the parent-level token within SP_1 is represented by $tokenAt_{SP_1}$. Determining whether all child-level tokens are consumed is formulated as $\bigwedge_{i=1}^{n_1} tokenConsumedBy_{E_i}$.

Although there are ten sorts of catch intermediate events, merely seven of them are capable of interrupting an activity. These encompass message, timer, error, conditional, signal, multiple and cancel. We now consider the first six kinds of catch intermediate events as depicted in Figure 10. The cancel intermediate event is discussed later in a definition pertaining to the behavioural semantics of a transaction.

Attaching a catch intermediate event E_1 to the boundary of an activity A_1 signifies that the activity A_1 is interruptible (Figure 10). Whenever a token reaches the interruptible activity A_1, an additional token is produced and placed on the catch intermediate event E_1. The tokens, which are located at the activity A_1 and the catch intermediate event E_1, are sent along the outgoing sequence flow and consumed respectively when the execution of the activity A_1 completes prior to the occurrence of the catch intermediate event E_1. If the happening of the catch intermediate event E_1 is earlier than the termination of the activity A_1, the activity A_1 ceases immediately. The additional token exits the catch intermediate event E_1 and traverses the outgoing sequence flow. Simultaneously, the token resided in the activity A_1 is consumed.

We define Γ_{X} = {Msg, Timer, Err, Cond, Sign, Multi} and Γ_{SP} = {Embed, Reuse, Ref}. The sets of activities and transactions are given by $F_{\mathrm{A}} = F_{\mathrm{T}} \cup \bigcup_{\sigma \in \Gamma_{\mathrm{SP}}} F_{\mathrm{SP}}^{\sigma}$ and $S_{\mathrm{TX}} = \{x | x \in \bigcup_{\sigma \in \Gamma_{\mathrm{SP}}} F_{\mathrm{SP}}^{\sigma} \wedge \Phi_{\mathrm{IsTX}}(x) = \mathit{true}\}$. We let $\Phi_{\mathrm{IE}}^{\mathrm{Bdy}[-\mathrm{TX}]} : F_{\mathrm{A}} \setminus S_{\mathrm{TX}} \rightarrow 2^{\bigcup_{\sigma \in \Gamma_{\mathrm{X}} \cup \{\mathrm{Cmpen}\}} F_{\mathrm{IE}}^{\sigma}}$ be a function returning the set of intermediate events attached to the boundary of an activity that is not a transaction.

Definition 17 (Exception): Suppose $A_1 \in F_{\mathrm{T}}$, $E_1 \in \bigcup_{\sigma \in \Gamma_{\mathrm{X}}} F_{\mathrm{IE}}^{\sigma}$, $\Phi_{\mathrm{IE}}^{\mathrm{Bdy}[-\mathrm{TX}]}(A_1) = \{E_1\}$, FO_1, FO_2, $FO_3 \in S_{\mathrm{F}}$, (FO_1, A_1), (A_1, FO_2), $(E_1, FO_3) \in C_{\mathrm{SF}}$ and $TE_k \in F_{\mathrm{EE}}^{\mathrm{Term}}$ for $k = 1, \ldots, m$. The interruption of a task A_1 caused by an intermediate event E_1 is represented as:

$$\begin{aligned}
\mathrm{G}\Bigg(&\Big(\big(tokenAt_{FO_1,A_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}\big) \rightarrow \\
&\quad \mathrm{X}(tokenAt_{A_1} \wedge tokenAt_{E_1} \wedge \neg tokenAt_{FO_1,A_1})\Big) \wedge \\
&\Big(\big(tokenAt_{A_1} \wedge \neg catchEvent_{E_1} \wedge \\
&\quad \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}\big) \rightarrow \\
&\quad \mathrm{F}(tokenAt_{A_1,FO_2} \wedge \neg tokenAt_{A_1} \wedge \neg tokenAt_{E_1})\Big) \wedge \\
&\Big(\big(tokenAt_{E_1} \wedge catchEvent_{E_1} \wedge \\
&\quad \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}\big) \rightarrow \\
&\quad \mathrm{X}(tokenAt_{E_1,FO_3} \wedge \neg tokenAt_{E_1} \wedge \neg tokenAt_{A_1})\Big)\Bigg).
\end{aligned}$$

The normal flow and exception flow are specified, respectively, in LTL by:

$$\begin{aligned}
&\Big(\big(tokenAt_{A_1} \wedge \neg catchEvent_{E_1} \wedge \\
&\quad \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}\big) \rightarrow
\end{aligned}$$

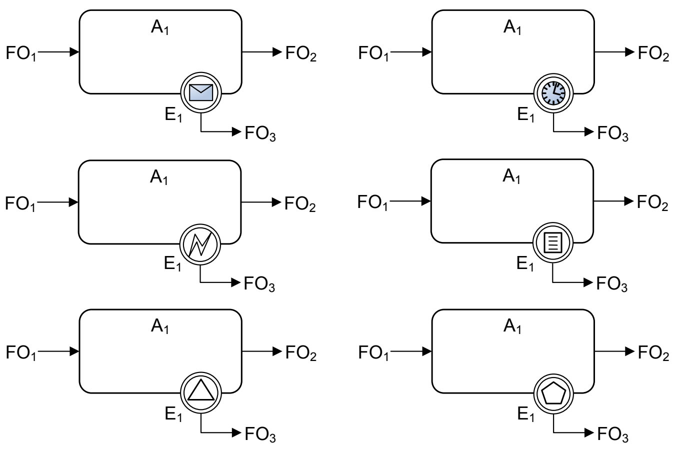

Fig. 10. A task A_1 interrupted by an intermediate event E_1

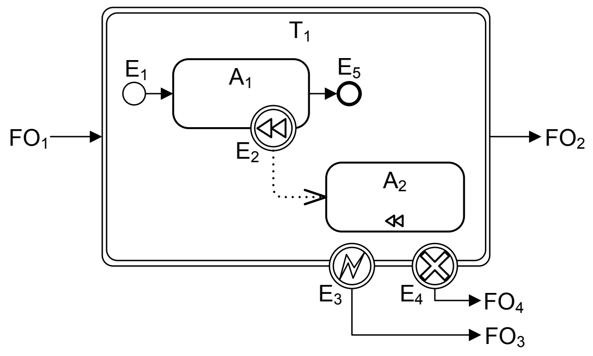

Fig. 11. Error and cancellation in a transaction T_1

$$\mathrm{F}(tokenAt_{A_1,FO_2} \wedge \neg tokenAt_{A_1} \wedge \neg tokenAt_{E_1})\Big)$$

and

$$\Big((tokenAt_{E_1} \wedge catchEvent_{E_1} \wedge \bigwedge_{k=1}^{m} \neg tokenConsumedBy_{TE_k}) \rightarrow$$
$$\mathrm{X}(tokenAt_{E_1,FO_3} \wedge \neg tokenAt_{E_1} \wedge \neg tokenAt_{A_1})\Big)$$

as stated in Definition 17.

The receipt of an error intermediate event E_3 leads to the immediate termination of the transaction T_1 as shown in Figure 11. The token that is placed on E_3 moves down the outgoing sequence flow and no compensation activity is invoked.

The triggering of the cancel intermediate event E_4 results in the token resided in the none end event E_5 to move in reverse direction. When the token arrives at the activity A_1, another token is generated and put on the compensation intermediate event E_2. The newly created token traverses the outgoing directed association. Upon receipt of the token, the compensation activity A_2 is executed. On completion of the compensation activity A_2, the token situated at A_1 keeps on moving backward until it reaches the none start event E_1. The arrival of the token at E_1 symbolizes the transaction T_1 is cancelled and causes the token located at E_4 to move down the outgoing sequence flow.

Let $\Gamma_{\mathrm{NL}} = \{\mathrm{None}, \mathrm{Link}\}$, $S_{\mathrm{T}}^{\mathrm{M}} = \{\{M_{\mathrm{L}}\}, \{M_{\mathrm{MI}}\}, \{M_{\mathrm{C}}\}, \{M_{\mathrm{L}}, M_{\mathrm{C}}\}, \{M_{\mathrm{MI}}, M_{\mathrm{C}}\}\}$, C_{DA} be a set of directed associations, $\Phi_{\mathrm{IE}}^{\mathrm{Bdy[TX]}} : S_{\mathrm{TX}} \rightarrow 2^{\bigcup_{\sigma\in\Gamma_{\mathrm{IE}}\setminus\Gamma_{\mathrm{NL}}} F_{\mathrm{IE}}^{\sigma}}$ be a function that returns the set of intermediate events attached to the boundary of a transaction and $\Phi_{\mathrm{TM}} : F_{\mathrm{T}} \rightarrow S_{\mathrm{T}}^{\mathrm{M}}$ be a function that returns the set of markers for a task.

Definition 18 (Transaction): Suppose $T_1 \in \bigcup_{\sigma\in\Gamma_{\mathrm{SP}}} F_{\mathrm{SP}}^{\sigma}$, $\Phi_{\mathrm{IsTX}}(T_1) = true$, $E_1 \in F_{\mathrm{SE}}^{\mathrm{None}}$, $E_2 \in F_{\mathrm{IE}}^{\mathrm{Cmpen}}$, $E_3 \in F_{\mathrm{IE}}^{\mathrm{Err}}$, $E_4 \in F_{\mathrm{IE}}^{\mathrm{Cncl}}$, $E_5 \in F_{\mathrm{EE}}^{\overline{\mathrm{None}}}$, A_1, $A_2 \in F_{\mathrm{T}}$, FO_1, FO_2, FO_3, $FO_4 \in S_{\mathrm{F}}$, $\Phi_{\mathrm{IE}}^{\mathrm{Bdy[TX]}}(T_1) = \{E_3, E_4\}$, $\Phi_{\mathrm{IE}}^{\mathrm{Bdy[-TX]}}(A_1) = \{E_2\}$, $\Phi_{\mathrm{TM}}(A_2) = \{M_{\mathrm{C}}\}$, (FO_1, T_1), (E_1, A_1), (A_1, E_5), (T_1, FO_2), (E_3, FO_3), $(E_4, FO_4) \in C_{\mathrm{SF}}$ and $(E_2, A_2) \in C_{\mathrm{DA}}$. The interruption of the transaction T_1 by the error intermediate event E_3 is formalized by:

$$\mathrm{G}\Big((tokenAt_{E_3} \wedge catchEvent_{E_3}) \rightarrow$$
$$\mathrm{X}(tokenAt_{E_3,FO_3} \wedge \neg tokenAt_{T_1} \wedge \neg tokenAt_{E_1} \wedge$$
$$\neg tokenAt_{E_1,A_1} \wedge \neg tokenAt_{A_1} \wedge$$
$$\neg tokenAt_{A_1,E_5} \wedge \neg tokenAt_{E_5} \wedge \neg tokenAt_{E_3} \wedge$$
$$\neg tokenAt_{E_4})\Big).$$

The cancellation of the transaction T_1 is modelled as:

$$\mathrm{G}\Big(\Big((tokenAt_{E_4} \wedge catchEvent_{E_4} \wedge tokenAt_{E_5}) \rightarrow$$
$$\mathrm{X}(tokenAt_{A_1,E_5} \wedge \neg tokenAt_{E_5})\Big) \wedge$$
$$\Big((tokenAt_{E_4} \wedge catchEvent_{E_4} \wedge tokenAt_{A_1,E_5}) \rightarrow$$
$$\mathrm{X}(tokenAt_{A_1} \wedge tokenAt_{E_2} \wedge \neg tokenAt_{A_1,E_5})\Big) \wedge$$
$$\Big((tokenAt_{E_4} \wedge catchEvent_{E_4} \wedge tokenAt_{E_2} \wedge$$
$$catchEvent_{E_2}) \rightarrow$$
$$\mathrm{X}(tokenAt_{E_2,A_2} \wedge \neg tokenAt_{E_2})\Big) \wedge$$
$$\Big((tokenAt_{E_4} \wedge catchEvent_{E_4} \wedge tokenAt_{E_2,A_2}) \rightarrow$$
$$\mathrm{X}(tokenAt_{A_2} \wedge \neg tokenAt_{E_2,A_2})\Big) \wedge$$
$$\Big((tokenAt_{E_4} \wedge catchEvent_{E_4} \wedge tokenAt_{A_2}) \rightarrow$$
$$\mathrm{F}(tokenAt_{E_1,A_1} \wedge \neg tokenAt_{A_1} \wedge \neg tokenAt_{A_2})\Big) \wedge$$
$$\Big((tokenAt_{E_4} \wedge catchEvent_{E_4} \wedge tokenAt_{E_1,A_1}) \rightarrow$$

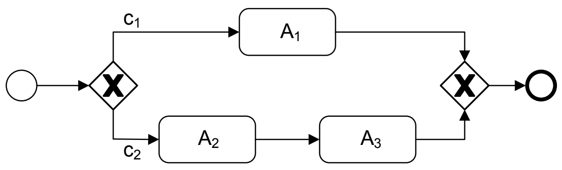

Fig. 12. BPMN process

$$\mathrm{X}(tokenAt_{E_1} \wedge \neg tokenAt_{E_1,A_1})\Big)\wedge$$
$$\Big((tokenAt_{E_4} \wedge catchEvent_{E_4} \wedge tokenAt_{E_1}) \rightarrow$$
$$\mathrm{X}(tokenAt_{E_4,FO_4} \wedge \neg tokenAt_{E_1} \wedge \neg tokenAt_{E_4})\Big)\Big).$$

In Definition 18, the backward movement of the token from E_5 to the sequence flow connecting A_1 and E_5 is expressed as:

$$\Big((tokenAt_{E_4} \wedge catchEvent_{E_4} \wedge tokenAt_{E_5}) \rightarrow$$
$$\mathrm{X}(tokenAt_{A_1,E_5} \wedge \neg tokenAt_{E_5})\Big).$$

The token on E_4 ultimately travels down the outgoing sequence flow is denoted in LTL by:

$$\Big((tokenAt_{E_4} \wedge catchEvent_{E_4} \wedge tokenAt_{E_1}) \rightarrow$$
$$\mathrm{X}(tokenAt_{E_4,FO_4} \wedge \neg tokenAt_{E_1} \wedge \neg tokenAt_{E_4})\Big).$$

VI. Analysis of BPMN Workflows

To illustrate the utilization of the defined semantics for a rigorous analysis of BPMN models, we consider the BPMN process in Figure 12. The none start event, data-based exclusive decision gateway, data-based exclusive merge gateway and none end event are labelled with E_1, G_1, G_2 and E_2, respectively. Applying Definition 11, the execution semantics of the none start event E_1 is represented as:

$$\mathrm{G}\Big((catchEvent_{E_1} \rightarrow \mathrm{X}tokenAt_{E_1})\wedge$$
$$\Big(tokenAt_{E_1} \rightarrow \mathrm{X}(tokenAt_{E_1,G_1} \wedge \neg tokenAt_{E_1})\Big)\Big).$$

The temporal formula infers the following reachability property:

$$\mathrm{G}(catchEvent_{E_1} \rightarrow \mathrm{F}tokenAt_{E_1,G_1}).$$

We assume that c_1 evaluates to true. The behaviour of the data-based exclusive decision gateway G_1 and the task A_1 is modelled in LTL according to Definitions 3 and 15 by:

$$\mathrm{G}\Big((tokenAt_{E_1,G_1} \wedge c_1) \rightarrow$$
$$\mathrm{X}(tokenAt_{G_1,A_1} \wedge \neg tokenAt_{E_1,G_1})\Big)$$

and

$$\mathrm{G}\Big(\Big(tokenAt_{G_1,A_1} \rightarrow \mathrm{X}(tokenAt_{A_1} \wedge \neg tokenAt_{G_1,A_1})\Big)\wedge$$
$$\Big(tokenAt_{A_1} \rightarrow \mathrm{F}(tokenAt_{A_1,G_2})\Big)\Big).$$

The LTL specifications logically imply the following conditional reachability property:

$$\mathrm{G}\Big((tokenAt_{E_1,G_1} \wedge c_1) \rightarrow \mathrm{F}tokenAt_{A_1}\Big).$$

Literally, the two derived formulas mean that (i) a token will eventually arrive at the sequence flow between E_1 and G_1 if a start event trigger occurs; and (ii) the task A_1 will eventually reach an execution state if a token is received and the condition c_1 is met. Likewise, the behavioural semantics of the data-based exclusive merge gateway G_2 and the none end event E_2 is encoded through the use of Definitions 4 and 14 as:

$$\mathrm{G}\Big(\Big(tokenAt_{A_1,G_2} \rightarrow \mathrm{X}(tokenAt_{G_2,E_2} \wedge \neg tokenAt_{A_1,G_2})\Big)\wedge$$
$$\Big(tokenAt_{A_3,G_2} \rightarrow \mathrm{X}(tokenAt_{G_2,E_2} \wedge \neg tokenAt_{A_3,G_2})\Big)\Big)$$

and

$$\mathrm{G}\Big(\Big(tokenAt_{G_2,E_2} \rightarrow \mathrm{X}(tokenAt_{E_2} \wedge \neg tokenAt_{G_2,E_2})\Big)\wedge$$
$$\Big(tokenAt_{E_2} \rightarrow \mathrm{X}(tokenConsumedBy_{E_2} \wedge \neg tokenAt_{E_2})\Big)\Big).$$

Combining the LTL formulas based on Definitions 15, 4 and 14, we obtain another reachability property:

$$\mathrm{G}(tokenAt_{A_1} \rightarrow \mathrm{F}tokenAt_{E_2}).$$

The formula is interpreted as a token placed on the task A_1 will eventually arrive at the none end event E_2.

In a similar way, we assume that c_2 is true. The encoding of G_1 (Definition 3) is given by:

$$\mathrm{G}\Big((tokenAt_{E_1,G_1} \wedge c_2) \rightarrow \mathrm{X}(tokenAt_{G_1,A_2} \wedge \neg tokenAt_{E_1,G_1})\Big).$$

The behaviour of tasks A_2 and A_3 is expressed in LTL (Definition 15) as shown below:

$$\mathrm{G}\Big(\Big(tokenAt_{G_1,A_2} \rightarrow \mathrm{X}(tokenAt_{A_2} \wedge \neg tokenAt_{G_1,A_2})\Big)\wedge$$
$$\Big(tokenAt_{A_2} \rightarrow \mathrm{F}(tokenAt_{A_2,A_3})\Big)\Big)$$

$$\mathrm{G}\Big(\Big(tokenAt_{A_2,A_3} \rightarrow \mathrm{X}(tokenAt_{A_3} \wedge \neg tokenAt_{A_2,A_3})\Big)\wedge$$
$$\Big(tokenAt_{A_3} \rightarrow \mathrm{F}(tokenAt_{A_3,G_2})\Big)\Big).$$

Consider these three formulas and the LTL expressions of G_2 and E_2, the following conditional reachability property is obtained:

$$\mathrm{G}\Big((tokenAt_{E_1,G_1} \wedge c_2) \rightarrow \mathrm{F}tokenAt_{E_2}\Big).$$

In addition to the reachability properties, the other soundness criterion for the BPMN process in Figure 12 is the fulfillment of liveness properties. Based on the LTL specifications of the BPMN process, the following liveness formula is yielded:

$$\mathrm{G}\Big(catchEvent_{E_1} \rightarrow \mathrm{F}tokenAt_{E_2}\Big).$$

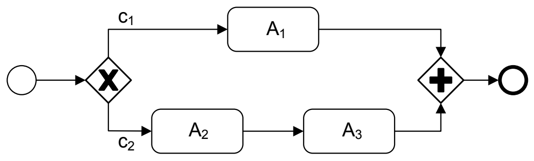

Fig. 13. BPMN process with a deadlock

Stated in words, any occurrence of a start event trigger will eventually lead to the happening of the none end event.

The satisfaction of the liveness property ascertains that every process instance of Figure 12 will finally terminate. The execution semantics of E_2 guarantees the consumption of the token whenever the process instance ends. Additionally, every graphical construct in the BPMN process has the possibility of being executed in accordance to the reachability properties. Hence, we can conclude that the BPMN model in Figure 12 is sound.

Through the substitution of the data-based exclusive merge gateway in Figure 12 by a parallel join gateway, we get another BPMN process as depicted in Figure 13. To check the validity of the BPMN process, the parallel join gateway is labelled with G_3. According to Definition 2, the behaviour semantics of G_3 is specified in LTL as:

$$\begin{aligned} \mathrm{G}\Big(& (tokenAt_{A_1,G_3} \wedge tokenAt_{A_3,G_3}) \rightarrow \\ & \mathrm{X}(tokenAt_{G_3,E_2} \wedge \\ & \neg tokenAt_{A_1,G_3} \wedge \neg tokenAt_{A_3,G_3})\Big). \end{aligned}$$

Since either c_1 or c_2 holds, the parallel join gateway G_3 is never executed. The BPMN process in Figure 13 does not satisfy the liveness property defined earlier owing to the presence of deadlock. Consequently, it is regarded as an unsound BPMN model.

VII. Conclusions

An unambiguous definition of the execution semantics of BPMN is of paramount importance. The absence of a precise semantics hampers (i) the semantic analysis of BPMN; (ii) the verification and reasoning on BPMN models; and (iii) the equivalence checking of BPMN diagrams. This paper has bridged the gap by formalizing the token-based semantics of BPMN in terms of LTL. The behavioural mechanism of each graphical construct has been codified using one or more LTL formulas. Our work is regarded as a supplement to the official BPMN documentation. It contributes to both the theoretical and practical facets of BPMN.

A promising line of inquiry is to further explore the applicability of the formal semantics. Specifically, we plan to construct a workflow engine that is based on the proposed semantic foundation. Another possible direction is to adapt our LTL-based semantics to BPMN 2.0 [47]. The BPMN 2.0, like BPMN 1.2, does not offer a well-founded execution semantics using mathematical techniques. Extending our work to BPMN 2.0 is relatively easy since the operational semantics of BPMN 2.0 is considered as a variant of the one given in BPMN 1.2.

References

[1] OMG, “Business process modeling notation, v1.1,” Jan. 2008, http://www.bpmn.org/; accessed January 7, 2009.
[2] S. White and D. Miers, *BPMN Modeling and Reference Guide*. Future Strategies Inc., 2008.
[3] OMG, “Business process modeling notation, v1.2,” Jan. 2009, http://www.bpmn.org/; accessed February 13, 2010.
[4] O. M. Group, “OMG unified modeling language^{TM} (OMG UML), superstructure version 2.2,” Feb. 2009, http://www.omg.org/spec/UML/2.2/Superstructure; accessed Feb 13, 2010.
[5] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.
[6] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*, 2nd ed. Cambridge University Press, 2004.
[7] I. Raedts, M. Petkovic, Y. Usenko, J. van der Werf, J. Groote, and L. Somers, “Transformation of BPMN models for behaviour analysis,” in *MSVVEIS 2007*, 2007, pp. 126–137.
[8] R. Dijkman, M. Dumas, and C. Ouyang, “Semantics and analysis of business process models in BPMN,” *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.
[9] A. Ratzer, L. Wells, H. Lassen, M. Laursen, J. Qvortrup, M. Stissing, M. Westergaard, S. Christensen, and K. Jensen, “CPN tools for editing, simulating, and analysing coloured petri nets.” in *ICATPN 2003*, ser. LNCS 2679. Springer-Verlag, 2003, pp. 450–462.
[10] C. Ou-Yang and Y. Lin, “BPMN-based business process model feasibility analysis: A Petri Net approach,” *International Journal of Production Research*, vol. 46, no. 14, pp. 3763–3781, 2008.
[11] R. Milner, J. Parrow, and D. Walker, “A calculus of mobile process (Parts I and II),” *Information and Computation*, vol. 100, pp. 1–77, 1992.
[12] R. Milner, “The polyadic π-calculus: A tutorial,” in *Logic and Algebra of Specification, Proceedings of International NATO Summer School*, vol. 94. Springer-Verlag, 1993, pp. 203–246.
[13] C. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.
[14] A. Bog, *Introduction to PiVizTool*, Hasso Plattner Institute, University of Potsdam, 2006, http://bpt.hpi.uni-potsdam.de/pub/Piworkflow/Simulator/piviztool-intro.pdf; accessed January 13, 2009.
[15] ——, “A visual environment for the simulation of business processes based on the pi-calculus,” Master’s thesis, Hasso Plattner Institute, University of Potsdam, 2006, http://bpt.hpi.uni-potsdam.de/pub/Public/FrankPuhlmann/AnjaBogThesisFinal.pdf; accessed January 13, 2009.
[16] A. Bog and F. Puhlmann, “A tool for the simulation of π-calculus systems,” in *Open.BPM 2006: Geschäftsprozessmanagement mit Open Source-Technologien*, 2006, http://bpt.hpi.uni-potsdam.de/pub/Public/FrankPuhlmann/PiSimulator_openBPM.pdf; accessed January 9, 2009.
[17] A. Bog, F. Puhlmann, and M. Weske, “The PiVizTool: Simulating choreographies with dynamic binding,” in *Demo Session of the 5th International Conference on Business Process Management*, 2007, http://bpt.hpi.uni-potsdam.de/pub/Public/FrankPuhlmann/bpm2007-piviztool.pdf; accessed February 17, 2008.
[18] F. Puhlmann, “Soundness verification of business processes specified in the pi-calculus,” in *CoopIS 2007*, ser. LNCS 4803, 2007, pp. 6–23.
[19] S. Briais, *The ABC User’s Guide*, 2005, http://lamp.epfl.ch/~sbriais/abc/abc_ug.pdf; accessed February 17, 2008.
[20] P. Wong and J. Gibbons, “A process semantics for BPMN,” in *Proceedings of the 10th International Conference on Formal Engineering Methods*, ser. LNCS 5256, 2008, pp. 355–374.
[21] ——, “Verifying business process compatibility,” in *Proceedings of the 8th International Conference on Quality Software*, 2008, pp. 126–131.
[22] F. S. E. Ltd., *Failures-Divergence Refinement: FDR2 User Manual*, May 2003, http://www.fsel.com/fdr2_download.html; accessed January 20, 2005.
[23] P. Wong and J. Gibbons, “A relative timed semantics for BPMN,” *Electronic Notes in Theoretical Computer Science*, vol. 229, no. 2, pp. 59–75, 2009.
[24] ——, “Formalisations and applications of BPMN,” *Science of Computer Programming*, 2009.
[25] R. Cavada, A. Cimatti, E. Olivetti, M. Pistore, and M. Roveri, *NuSMV 2.1 User Manual*, 2002, http://nusmv.irst.itc.it; accessed January 20, 2005.
[26] V. Lam, “Formal analysis of BPMN models: A NuSMV-based approach,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 20, no. 7, pp. 987–1023, 2010.
[27] M. Dumas, A. Grosskopf, T. Hettel, and M. Wynn, “Semantics of standard process models with OR-joins,” in *OTM 2007*, ser. LNCS 4803, 2007, pp. 41–58.

[28] E. Börger and B. Thalheim, "A method for verifiable and validatable business process modeling," in *Advances in Software Engineering, Lipari Summer School 2007*, ser. LNCS 5316, 2008, pp. 59–115.

[29] A. Grosskopf, "xBPMN: Formal control flow specification of a BPMN-based process execution language," Master's thesis, Hasso Plattner Institute, University of Potsdam, 2007, http://www.myhpi.de/~alexander.grosskopf/xBPMN_thesis.pdf; accessed November 14, 2010.

[30] V. Lam, "Equivalences of BPMN processes," *Service Oriented Computing and Applications*, vol. 3, no. 3, pp. 189–204, 2009.

[31] D. Auer, V. Geist, and D. Draheim, "Extending BPMN with submit/response-style user interaction modeling," in *CEC 2009*, 2009, pp. 368–374.

[32] S. Mazanek and M. Hanus, "Constructing a bidirectional transformation between BPMN and BPEL with a functional logic programming language," *Journal of Visual Languages and Computing*, vol. 22, no. 1, pp. 66–89, 2011.

[33] L. Sánchez-González, F. Ruiz, F. García, and J. Cardoso, "Towards thresholds of control flow complexity measures for BPMN models," in *SAC '11*, 2011, pp. 1445–1450.

[34] O. Kopp, F. Leymann, D. Schumm, and T. Unger, "On BPMN process fragment auto-completion," in *ZEUS 2011*, 2011, pp. 58–64.

[35] R. Müller and A. Rogge-Solti, "BPMN for healthcare processes," in *ZEUS 2011*, 2011, pp. 65–72.

[36] F. Maggi, M. Montali, M. Westergaard, and W. van der Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," in *BPM 2011*, ser. LNCS 6896, 2011, pp. 132–147.

[37] M. Gambini, M. La Rosa, S. Migliorini, and A. ter Hofstede, "Automated error correction of business process models," in *BPM 2011*, ser. LNCS 6896, 2011, pp. 148–165.

[38] R. Jagadeesh Chandra Bose and W. van der Aalst, "Abstractions in process mining: A taxonomy of patterns," in *BPM 2009*, ser. LNCS 5701, 2009, pp. 159–175.

[39] J. Carmona, J. Cortadella, and M. Kishinevsky, "Divide-and-conquer strategies for process mining," in *BPM 2009*, ser. LNCS 5701, 2009, pp. 327–343.

[40] R. Jagadeesh Chandra Bose and W. van der Aalst, "Trace alignment in process mining: Opportunities for process diagnostics," in *BPM 2010*, ser. LNCS 6336, 2010, pp. 227–242.

[41] D. Fahland and W. van der Aalst, "Simplifying mined process models: An approach based on unfoldings," in *BPM 2011*, ser. LNCS 6896, 2011, pp. 362–378.

[42] M. Pesic, H. Schonenberg, and W. van der Aalst, "Declarative workflow," in *Modern Business Process Automation*, 2010, pp. 175–201.

[43] M. Westergaard, "Better algorithms for analyzing and enacting declarative workflow languages using LTL," in *BPM 2011*, ser. LNCS 6896, 2011, pp. 83–98.

[44] R. Khalaf and F. Leymann, "Coordination for fragmented loops and scopes in a distributed business process," in *BPM 2010*, ser. LNCS 6336, 2010, pp. 178–194.

[45] V. Lam, "Foundation for equivalences of BPMN models," submitted for publication.

[46] H. Völzer, "A new semantics for the inclusive converging gateway in safe processes," in *BPM 2010*, ser. LNCS 6336, 2010, pp. 294–309.

[47] OMG, "Business Process Model and Notation (BPMN), version 2.0," Jan. 2011, http://www.omg.org/spec/BPMN/2.0; accessed May 23, 2011.