A First-Order Logic Semantics for Communication-Parametric BPMN Collaborations*

Sara Houhou[1,2], Souheib Baarir[1,3], Pascal Poizat[1,3], and Philippe Quéinnec[4]

[1] Sorbonne Université, CNRS, LIP6, F-75005, Paris, France
[2] Biskra University, LINFI Laboratory, Biskra, Algeria
[3] Université Paris Lumières, Université Paris Nanterre, F-92000, Nanterre, France
[4] IRIT - Université de Toulouse, F-31000 Toulouse, France
{sara.houhou, souheib.baarir, pascal.poizat}@lip6.fr
philippe.queinnec@irit.fr

Abstract. BPMN is suitable to model not only intra-organization workflows but also inter-organization collaborations. There has been a great effort in providing a formal semantics for BPMN, and then in building verification tools on top of this semantics. However, communication aspects are often discarded in the literature. This is an issue since BPMN has gained interest outside its original scope, e.g., for the IoT, where the configuration of communication modes plays an important role. In this paper, we propose a formal semantics for a subset of BPMN, taking into account inter-process communication and parametric verification with reference to communication modes. As opposed to transformational approaches, that map BPMN into some formal model such as transition systems or Petri nets, we give a direct formalization in First-Order Logic that is then implemented in TLA$^+$ to enable formal verification. Our approach is tool supported. The tool, as well as the TLA$^+$ theories, and experiment models are available online.

Keywords: BPMN · Formal Semantics · Collaboration · Communication · Verification · TLA$^+$ · Tool.

1 Introduction

BPMN collaboration diagrams [1] provide an efficient way to describe how several business entities, each one with its own internal process, can interact with one another to reach objectives. The BPMN standard defines an execution semantics using natural language, that could be qualified as semi-formal. This leaves room for interpretation and hampers formal analysis of the models. This issue has been addressed in the last decade in different proposals for a formalization of the BPMN execution semantics (see Section 4), some of them with available

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

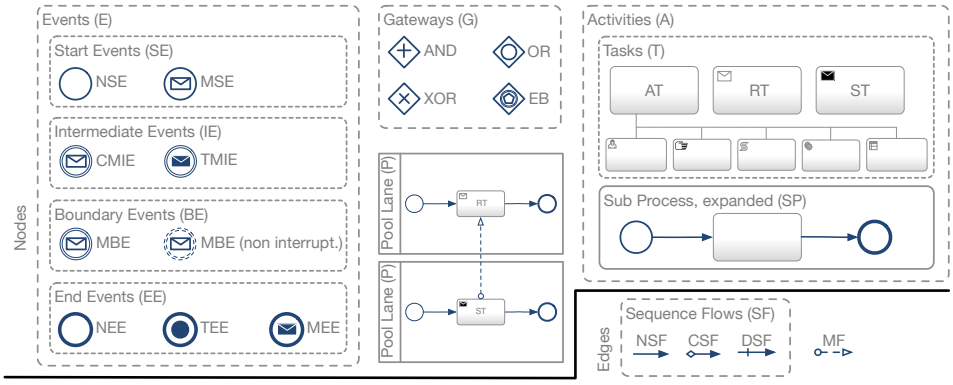

Fig. 1: BPMN subset being supported, with types used in the formalization (*e.g.*, MSE stands for Message Start Events and SE for Start Events). Due to lack of space, the support for boundary events is presented in [12].

tools. But these proposals often leave apart features related to communication. Meanwhile, BPMN is gaining interest as a modeling language for the Internet of Things (IoT) [3, 17], There, the communication between the nodes of the system, and the configuration of different communication modes, is an issue.

Contribution. The contribution of this paper is twofold:

1. provide a formalization of a subset of BPMN execution semantics that *supports interaction* and that is *parametric with reference to the properties of the communication* between participants,
2. support this formalization with *tools that automatically perform the verification of correctness properties* for BPMN collaboration models.

As far as (1.) is concerned, we chose to define a direct First-Order Logic (FOL) semantics for BPMN. Instead of using an intermediary formal model, *e.g.*, Petri nets or process algebra, this choice of a simple yet expressive framework enables one to get a formal semantics that is amenable to implementation in different formal frameworks while still being close to the semi-formal semantics of the standard (hence it can be related to it). We implement our FOL semantics in TLA$^+$ [16] as a set of TLA$^+$ theories. It supports six different communication models and is easily extensible. As far as the subset of BPMN is concerned, we have first based our choice on the analysis of the 825 BPMN processes available in the BIT process library, release 2009 [11], given in [14]. We have then taken more constructs into account, mainly relative to our focus on communication: creation and termination of processes using messages, message-related tasks and intermediary events, and event-based gateways. The whole subset of the notation that we support is given in Figure 1.

With respect to (2.), our approach relies on two steps. First, one uses the fbpmn tool that we have developed to get a TLA$^+$ representation of the model to verify. Then, one uses the TLC model-checker from the TLA$^+$ tool-suite to

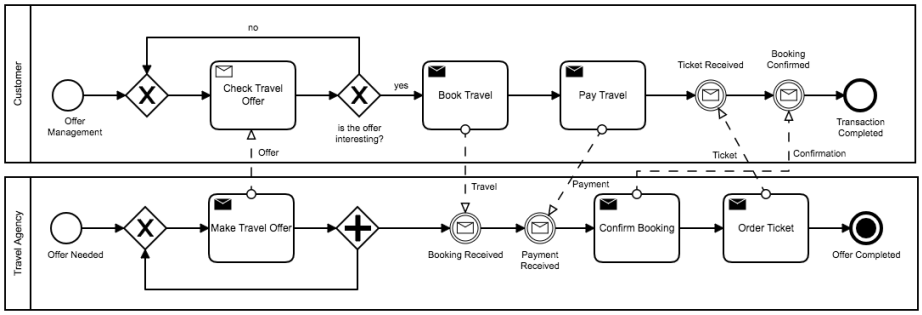

Fig. 2: Travel agency case study (slightly adapted from an example in [8]).

perform the verification. The properties of interest are encoded in the TLA$^+$ theories we have implemented. They include usual correctness properties for workflows as well as ones (proposed more recently [8]) that are more specific to BPMN. Both tools are open source and freely available online. Furthermore, the models we have used for evaluation in Section 3 are also available online. To get the tools and the models, see `https://github.com/pascalpoizat/fbpmn`.

Due to space considerations, we assume the reader has a basic knowledge of the BPMN notation. We refer to [1] if this is not the case.

Case study. Figure 2 presents the case study we use to illustrate some definitions and to perform verification. The outcomes of verification on more examples, including ones from the literature, are synthesized in Section 3.

The collaboration involves two participants: a customer and a travel agency. The agency sends offers to the client. The client may accept or decline the offers (loop using two exclusive gateways). Once it has accepted an offer, the client will not be ready to receive another one and it proceeds to the exchange part relative to the offer of interest. On the other side, the agency (through the use of a parallel gateway) is able both to send other offers and begin the exchange part: at the parallel gateway a token is generated both to get back to offer sending and to wait for the booking. This case study is interesting for several reasons. First, due to the agency behavior, the collaboration is possibly unsafe: one can have an unbounded number of tokens on the right of the parallel gateway. Second, observe that the partners do not agree on the order of confirmation *wrt.* ticket reception. Depending on the communication model, this may cause a deadlock.

Overview. The formal part of the paper is developed in Section 2, with Subsections 2.1 and 2.2, respectively addressing the presentation of the model underlying the semantics, and then the semantics itself. The implementation of the semantics in TLA$^+$, verification, and evaluation are then presented in Section 3. This section also includes a short introduction to the TLA$^+$ language and verification framework. Related work is given in Section 4, and we end with conclusions and perspectives in Section 5.

2 Formal Semantics

In this section, we first present the model on which we base the definition of the communication-parametric semantics for BPMN collaborations. This model is used to represent collaborations as typed graphs. In a second step, we present the semantic itself. It follows the "token game" of the standard [1, Ch. 13], with a notion of state that evolves with activation and completion of graph nodes.

2.1 A Typed Graph Representation of BPMN Collaborations

In our work, a BPMN model is seen as a typed graph (Def. 1), where nodes and edges are associated to types corresponding to the BPMN syntax (see Fig. 1): $T_{Nodes} = \{AT, RT, ST, SP, NSE, MSE, CMIE, TMIE, NEE, TEE, MEE, AND, OR, XOR, EB, P\}$ and $T_{Edges} = \{NSF, CSF, DSF, MF\}$. The hierarchical structure of collaborations, with processes and sub-processes is dealt with by using specific types for nodes, P and SP, and a relation denoting containment, R. From our example, Figure 2, we would then have two nodes of type P (say n_1 for Customer, n_2 for Travel Agency) with their respective contents related by R, *e.g.*, a node n_3 for Make Travel Offer of type ST with $n_3 \in R(n_2)$.

Definition 1 (BPMN Graph). *A BPMN graph is a tuple* $\widehat{G}$= *(N, E,* $\mathbb{M}$*,* cat_N*,* cat_E*,* $source, target$*, R,* msg_t*) where 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,* $source/target : E \to N$ *give the source/target of an edge,* $R : N^{\{SP,P\}} \to 2^{N \cup E}$ *gives the set of nodes and edges which are directly contained in a container (process or sub-process), and* $msg_t : E^{MF} \to \mathbb{M}$ *gives the message associated to a message flow.*

Notation. We use 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, we may write N^t instead of $N^{\{t\}}$, *e.g.*, N^{NSE} instead of $N^{\{NSE\}}$. We use for cat_E the same simplified notations as for cat_N.

We then define some auxiliary functions that will be used in the semantics.

Auxiliary functions. For a graph $\widehat{G}$= (N, E, $\mathbb{M}$, cat_N, cat_E, $source, target$, R, msg_t), we introduce the following auxiliary functions:

- $in/out : N \to 2^E$ give the incoming/outgoing edges of a node, $in(n) = \{e \in E \mid target(e) = n\}$ and $out(n) = \{e \in E \mid source(e) = n\}$.
- 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 $out^T(n) = out(n) \cap E^T$.
- $procOf : N \to N^P$ gives the container process of a given node, $procOf(n) = p$ if and only if $n \in R^+(p)$, with R^+ being the transitive closure of R.

It is desirable to enforce that models respect some well-formedness rules before performing verification. Due to a lack of space, we give the rules that we impose (most coming from [1]) for a BPMN graph to be well-formed in [12].

2.2 A FOL Semantics of BPMN Collaborations

In order to maintain traceability with the standard, we use a token-based approach. The movement of tokens is based on node types. We define an execution model based on two predicates (St, Ct) for each node type which correspond to, respectively, the enabling of the node to start its execution, and the enabling of the 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 definition of these predicates relies a notion of *state* of the BPMN Graph.

Definition 2 (State). *The state of a BPMN graph is given by a couple of functions* $s = (m_n, m_e)$, $m_n : N \to \mathbb{N}$ *and* $m_e : E \to \mathbb{N}$, *that associate a number of tokens to nodes (gateways are always 0) and edges, respectively. The set of all states of a BPMN graph is denoted by States.*

Definition 3 (Initial state). *The initial state of a BPMN graph, denoted by* $s_o = (m_{e0}, m_{n0})$, *associates a token to the* NSE *nodes of the processes, all the other nodes and edges being unmarked:* $\forall e \in E, m_{e0}(e) = 0$, *and* $\forall n \in N, m_{n0}(n) = 1$ *if* $\exists p \in N^P, n \in N^{NSE} \cap R(p)$, *and 0 otherwise.*

The properties of communication between two participants (process nodes) for a given type of message are abstracted with two predicates, *send* and *receive*. These predicates specify when a communication action is enabled and the effect of this communication. For instance, with FIFO asynchronous communication (`NetworkFifo`, Section 3.3), messages must be delivered in the order they were sent. Thus $send(p_1, p_2, m)$ is always enabled and $receive(p_1, p_2, m)$ is true only if m is the oldest message and thus the next one to be delivered. Observe that the value of these predicates evolve as the processes send and receive messages.

Definition 4 (Communication Model). *The communication model is characterized by two predicates* $send/receive : N^P \times N^P \times \mathbb{M} \to Bool$.

The formal execution semantics is given in Tables 1 and 2. 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. $\triangle$ is a predicate that denotes marking equality but for nodes and edges given as parameter, $\triangle(X)$ means "*nothing changes but for* X": $\triangle(X) \stackrel{def}{\equiv} \forall n \in N \setminus X, m'_n(n) = m_n(n) \wedge \forall e \in E \setminus X, m'_e(e) = m_e(n)$.

Starting and terminating. The behavior of an NSE node is defined only through completion: it consumes one token and generates one token on all its outgoing sequence flow edges. If it is the initial node of a process p, it activates it by generating a token on p. For a sub-process, it is the SP starting predicate that will perform activation (see below). The behavior of an NEE node is defined only through a starting predicate: it is enabled if it has at least one token on one of its incoming edges, that is added to the node. A TEE node is also defined only through a starting predicate: it is enabled if it has at least one token on one of its incoming edges, then it drops down all the remaining tokens of the process or sub-processes to which it belongs. The behavior of an activity node

Table 1: FOL semantics (part 1 – events)

Events	$n \in N^{NSE}$	$Ct(n) \stackrel{def}{\equiv} (m_n(n) \geq 1) \wedge (m'_n(n) = m_n(n) - 1)$ $\wedge (\forall e \in out^{SF}(n), (m'_e(e) = m_e(e) + 1))$ $\wedge ((\exists p \in N^P, n \in R(p) \wedge (m_n(p) = 0) \wedge (m'_n(p) = 1)$ $\wedge \triangle (\{n, p\} \cup out^{SF}(n)))$ $\vee (\exists p \in N^{SP}, n \in R(p) \wedge \triangle(\{n\} \cup out^{SF}(n))))$
	$n \in N^{MSE}$	$St(n) \stackrel{def}{\equiv} (m_n(n) = 0) \wedge (m'_n(n) = 1)$ $\wedge (\exists e \in in^{MF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge receive(procOf(source(e)), procOf(n), msg_t(e))$ $\wedge \triangle (\{n, e\}))$
		$Ct(n) \stackrel{def}{\equiv} (m_n(n) = 1) \wedge (m'_n(n) = m_n(n) - 1)$ $\wedge (\exists p \in N^P, n \in R(p) \wedge (m_n(p) = 0) \wedge (m'_n(p) = 1)$ $\wedge (\forall e \in out^{SF}(n), (m'_e(e) = m_e(e) + 1))$ $\wedge \triangle (\{n, p\} \cup out^{SF}(n)))$
	$n \in N^{TMIE}$	$St(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (\exists e' \in out^{MF}(n), (m'_e(e') = m_e(e') + 1)$ $\wedge send(procOf(n), procOf(target(e')), msg_t(e'))$ $\wedge \forall e" \in out^{SF}(n), (m'_e(e") = m_e(e") + 1)$ $\wedge \triangle (\{e, e'\} \cup out^{SF}(n))))$
	$n \in N^{CMIE}$	$St(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (\exists e' \in in^{MF}(n), (m_e(e') \geq 1) \wedge (m'_e(e') = m_e(e') - 1)$ $\wedge receive(procOf(source(e)), procOf(n), msg_t(e'))$ $\wedge (\forall e" \in out^{SF}(n), (m'_e(e") = m_e(e") + 1))$ $\wedge \triangle (\{e, e'\} \cup out^{SF}(n))))$
	$n \in N^{NEE}$	$St(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n),\ (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (m'_n(n) = m_n(n) + 1) \wedge \triangle(\{n, e\}))$
	$\forall n \in N^{TEE}$	$St(n) \stackrel{def}{\equiv} (m'_n(n) = m_n(n) + 1)$ $\wedge (\exists e \in in^{SF}(n), (m_e(e) \geq 1))$ $\wedge (\exists p \in N^{\{P,SP\}}, n \in R(p)$ $\wedge (\forall nn \in ((R^+(p) \cap N) \setminus \{n\}), m'_n(nn) = 0)$ $\wedge (\forall ee \in (R^+(p) \cap E), m'_e(ee) = 0)$ $\wedge \triangle (R^+(p)))$
	$n \in N^{MEE}$	$St(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (\exists e' \in out^{MF}(n), send(procOf(n), procOf(target(e')), msg_t(e'))$ $\wedge (m'_e(e') = m_e(e') + 1) \wedge (m'_n(n) = m_n(n) + 1) \wedge \triangle(\{n, e, e'\})))$

Table 2: FOL Semantics (part 2 – gateways and activities)

Gateways	$n \in N^{XOR}$	$Ct(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (\exists e' \in out^{SF}(n), (m'_e(e') = m_e(e') + 1))$ $\wedge \triangle (\{e, e'\}))$
	$n \in N^{AND}$	$Ct(n) \stackrel{def}{\equiv} (\forall e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1))$ $\wedge (\forall e' \in out^{SF}(n), (m'_e(e') = m_e(e') + 1))$ $\wedge \triangle (in^{SF}(n) \cup out^{SF}(n))$
	$n \in N^{OR}$	$Ct(n) \stackrel{def}{\equiv} (In^{+}(n) \neq \emptyset) \wedge (\forall e \in In^{+}(n), (m'_e(e) = m_e(e) - 1))$ $\wedge (\forall ez \in In^{-}(n), \forall ee \in (PreE(n, ez) \setminus ignoreE(n)), (m_e(ee) = 0)$ $\wedge (\forall nn \in (PreN(n, ez) \setminus ignoreN(n)), (m_n(nn) = 0)))$ $\wedge ((\exists Outs \subset (out^{SF}(n) \cap E^{\{NSF, CSF\}}), (Outs \neq \emptyset)$ $\wedge (\forall e \in Outs, (m'_e(e) = m_e(e) + 1) \wedge \triangle (In^{+}(n) \cup Outs))$ $\vee (\exists e \in out^{DSF}(n), (m'_e(e) = m_e(e) + 1) \wedge \triangle (In^{+}(n) \cup \{e\})))$
	$n \in N^{EB}$	$Ct(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (\exists e' \in out^{SF}(n), \exists e" \in in^{MF}(target(e')), (m_e(e") \geq 1)$ $\wedge (m'_e(e') = m_e(e') + 1) \wedge \triangle (e, e')))$
Activities	$n \in N^{AT}$	$St(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (m'_n(n) = m_n(n) + 1) \wedge \triangle (\{n, e\}))$
		$Ct(n) \stackrel{def}{\equiv} (m_n(n) \geq 1) \wedge (m'_n(n) = m_n(n) - 1)$ $\wedge (\forall e \in out^{SF}(n), (m'_e(e) = m_e(e) + 1)) \wedge \triangle (\{n\} \cup out^{SF}(n))$
	$n \in N^{ST}$	$St(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (m'_n(n) = m_n(n) + 1) \wedge \triangle (\{n, e\}))$
		$Ct(n) \stackrel{def}{\equiv} (m_n(n) \geq 1) \wedge (m'_n(n) = m_n(n) - 1)$ $\wedge (\forall e \in out^{SF}(n), (m'_e(e) = m_e(e) + 1))$ $\wedge (\exists e' \in out^{MF}(n), send(procOf(n), procOf(target(e')), msg_t(e'))$ $\wedge (m'_e(e') = m_e(e') + 1) \wedge \triangle (\{n, e'\} \cup out^{SF}(n)))$
	$n \in N^{RT}$	$St(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (m'_n(n) = m_n(n) + 1) \wedge \triangle (\{n, e\}))$
		$Ct(n) \stackrel{def}{\equiv} (m_n(n) \geq 1) \wedge (m'_n(n) = m_n(n) - 1)$ $\wedge (\forall e \in out^{SF}(n), (m'_e(e) = m_e(e) + 1))$ $\wedge (\exists e' \in in^{MF}(n), (m_e(e') \geq 1) \wedge (m'_e(e') = m_e(e') - 1)$ $\wedge receive(procOf(source(e')), procOf(n), msg_t(e'))$ $\wedge \triangle (\{n, e'\} \cup out^{SF}(n)))$
	$n \in N^{SP}$	$St(n) \stackrel{def}{\equiv} (\exists e \in in^{SF}(n), (m_e(e) \geq 1) \wedge (m'_e(e) = m_e(e) - 1)$ $\wedge (m'_n(n) = m_n(n) + 1)$ $\wedge (\forall n_{se} \in (N^{NSE} \cap R(n)), (m'_n(n_{se}) = m_n(n_{se}) + 1))$ $\wedge \triangle (\{e, n\} \cup (N^{NSE} \cap R(n))))$
		$Ct(n) \stackrel{def}{\equiv} (m_n(n) \geq 1) \wedge (m'_n(n) = 0)$ $\wedge (\forall e \in R(n) \cap E, (m_e(e) = 0))$ $\wedge (\exists n_{ee} \in (N^{EE} \cap R(n)), (m_n(n_{ee}) \geq 1))$ $\wedge (\forall nn \in R(n) \cap N, (m_n(nn) \geq 1 \Rightarrow nn \in N^{EE}))$ $\wedge (\forall nn \in (R(n) \cap N^{EE}), (m'_n(nn) = 0))$ $\wedge (\forall e \in out^{SF}(n), (m'_e(e) = m_e(e) + 1))$ $\wedge \triangle (\{n\} \cup (R(n) \cap N^{EE}) \cup out^{SF}(n))$

AT is defined by a starting and a completion predicate. The node is started by the arrival of at least one token on one of its incoming edges. Completion is realized by adding one token on each of its outgoing edges. The behavior of a SP node extends the one of an AT node with some additional conditions (due to its compound architecture): when enabled, a sub-process adds a token to the start event it contains. It completes when an end event it contains has some tokens and none of its other nodes and edges are still active (*i.e.*, owning a token).

Communication. The communication elements (MSE, $TMIE$, $CMIE$, MEE, ST, RT) require additional conditions for starting and completing due to the presence of sending/reception behaviors. When enabled by a token on one of their incoming edges, $TMIE$, MEE, and ST send a message on their outgoing message flow and a token on all their outgoing sequence flows. MSE, $CMIE$, RT are ready to start if they have a message offer on one of their incoming message flows. They receive the message and produce tokens on their outgoing edges. MSE and RT have both starting and completion transitions while $CMIE$ is an instantaneous event with only a starting transition.

Gateways. Gateways are atomic and define only the completion behavior. An AND gateway is ready to complete if it has at least one token on all its incoming edges. It completes by removing one token on each of these edges, and producing one on all its outgoing edges. An XOR gateway is ready to complete if it has at least one token on one of its incoming edges. It completes by removing this token, and producing one on one of its outgoing edges, depending on conditions. Since we abstract from data, we proceed choosing non-deterministically the concerned edge. An EB gateway behaves as an XOR gateway in that it consumes and produces a token from and to only one edge. However, its completion relies on the presence of external message triggers. The outgoing edge activation depends on the enabledness of the $CMIE$ or RT which is the target of this edge. The activation of an OR gateway g is more complex [1, Chap. 13]. It is activated if (1) it has at least one token on one of its incoming edges, and (2) for each node or edge x such that there is a path – that does not pass through g – from x to an unmarked incoming edge of g, there must be also a path – that does not pass through g – from x to a marked incoming edge of g. This is illustrated in Figure 3 where gateway g cannot be activated since there is a path from marked e^{pre} to unmarked e^- and no path from e^{pre} to some marked e^+. The OR gateway completes by adding a token either to all its outgoing edges whose condition it true, or else to its default sequence flow edge. Again, since we abstract from data, we chose non-deterministically to add a token either to a combination (1 or more) of the outgoing non default edges, or to the default edge.

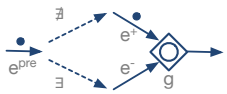

Fig. 3: Non activable inclusive gateway. It has to wait for the token on e^{pre}.

To formalize the semantics of an *OR* gateway we use several functions:

- $Pre_N : N \times E \to 2^N$ gives the predecessor nodes of an edge such that n^{pre} is in $Pre_N(n, e)$ if there is a path from n^{pre} to e that never visits n. Accordingly, $Pre_E : N \times E \to 2^E$ gives predecessor edges.
- $In^- : N \to E$ gives the unmarked incoming edges of a node, $In^-(n) = \{e^- \in in^{SF}(n) \mid m_e(e^-) = 0\}$.
- $In^+ : N \to E$ gives the marked incoming edges of a node, $In^+(n) = \{e^+ \in in^{SF}(n) \mid m_e(e^+) \geq 1\}$.
- $ignore_E : N \to 2^E$ gives the predecessor edges of the marked incoming edges of a given node: $ignore_E(n) \stackrel{def}{\equiv} \bigcup_{e^+ \in In^+(n)} Pre_E(n, e^+)$.
- $ignore_N : N \to 2^N$ gives the predecessor nodes of the marked incoming edges of a given node: $ignore_N(n) = \bigcup_{e^+ \in In^+(n)} Pre_N(n, e^+)$.

3 Implementation & Verification

In this section, we present our encoding of the FOL semantics in TLA$^+$. This allows one to easily parameter the properties of the communication, and to benefit from the efficient TLC model checker to automatically verify collaborations.

3.1 The TLA$^+$ Specification Language and Verification Framework

TLA$^+$ [16] is a formal specification language based on untyped Zermelo-Fraenkel set theory for specifying data structures, and on the temporal logic of actions (TLA) for specifying dynamic behaviors. TLA$^+$ allows one to specify symbolic transition systems with variables and *actions*. An action is a transition predicate between a state and a successor state. It is an arbitrary first-order predicate with quantifiers, set and arithmetic operators, and functions. In an action, x denotes the value of a variable x in the origin state, and x' denotes its value in the next state. Functions are primitive objects in TLA$^+$. The application of function f to an expression e is written as $f[e]$. The expression $[x \in X \mapsto e]$ denotes the function with domain X that maps any $x \in X$ to e. The expression $[f \text{ EXCEPT } ![e_1] = e_2]$ is a function that is equal to the function f except at point e_1, where its value is e_2. A system specification is usually a disjunction of actions. Fairness, usually expressed as a conjunction of weak or strong fairness on actions, or more generally as an LTL (Linear Temporal Logic) property, ensures progression. The TLA$^+$ toolbox, freely available at `http://lamport.azurewebsites.net/tla/tla.html`, contains the TLC model checker, the TLAPS proof assistant, and various other tools.

3.2 Encoding of FOL Semantics in TLA$^+$

The expression and action fragment of TLA$^+$ contains FOL, and the encoding of the semantics in TLA$^+$ is straightforward (459 lines of TLA$^+$ formulae). The resulting theories are available in [12] and at `https://github.com/pascalpoizat/fbpmn` under `theories/tla`.

Module `PWSTypes` defines the abstract constants that correspond to the node and edge types. Module `PWSDefs` specifies the constants that describe a BPMN graph (Definition 1): `Node` (for N), `Edge` (for E), `Message` (for $\mathbb{M}$), `CatN` (for cat_N), `CatE` (for cat_E), `ContainRel` (for R)... This module also defines auxiliary functions such as in^T. Module `PWSWellFormed` encodes the well-formedness predicates for BPMN graphs. Last, module `PWSSemantics` contains the semantics. It defines the variables for the marking: `nodemarks` ($\in [Node \to Nat]$) and `edgemarks` ($\in [Edge \to Nat]$). Then it contains a translation of the FOL formulas given in Tables 1 and 2. Each rule yields one TLA$^+$ action. For instance, the St predicate of the SP nodes becomes:

$$
\begin{array}{l}
subprocess\_start(n) \triangleq \\
\quad CatN[n] = SubProcess \\
\land\ \exists e \in intype(SeqFlowType, n) : edgemarks[e] \geq 1 \\
\qquad \land\ edgemarks' = [edgemarks \text{ EXCEPT } ![e] = edgemarks[e] - 1] \\
\land\ nodemarks' = [nn \in \text{DOMAIN } nodemarks \mapsto \\
\qquad \text{IF } nn = n \text{ THEN } 1 \\
\qquad \text{ELSE IF } CatN[nn] = StartEvent \land nn \in ContainRel[n] \text{ THEN } nodemarks[nn] + 1 \\
\qquad \text{ELSE } nodemarks[nn]] \\
\land\ Network!unchanged
\end{array}
$$

The $Next$ predicate specifies a possible transition between a starting state and a successor state. It is a disjunction of all the actions. The full specification is then, as usual in TLA$^+$, $Init \land \Box[Next]_{vars} \land Fairness$, where $Init$ specifies the initial state (Def. 3), and $\Box[Next]$ specifies that $Next$ (or stuttering) is verified along all the execution steps. In our case, $Fairness$ is a temporal property that ensures that any permanently enabled transition eventually occurs. This means that no process may progress forever while others are never allowed to do so if they can. Moreover, we include in the fairness that no choice is infinitely often ignored: if a XOR, OR, or EB gateway is included in a loop, the fairness forbids the infinite executions that never use some output edges.

3.3 Communication as a Parameter

One of the objectives of our FOL semantics is to be able to specify the communication behavior as a parameter of the verification. To achieve this, all operations related to communication are isolated in a `Network` module. This module is a proxy for several implementations that correspond to communication models with different properties, such as their delivery order.

We provide six communication models which differ in the order messages can be sent or received, and are all the possible point-to-point models when considering local ordering (per process) and global ordering (absolute time). They are formally defined in [4] and informally are: unordered (modelled by a bag of message), first-in-first-out between each couple of processes (modelled by an array of queues), fifo inbox (each process has an input queue where senders add their messages), fifo outbox (each process has an output queue from where receivers fetch messages), global fifo (a unique shared queue), and RSC (realizable with

synchronous communication, modelled as a unique message slot that forces alternation of send and receive tasks). If the collaboration is sound, no message is left in transit, and RSC and synchronous communication yield the same behaviors.

The state of the communication model is specified with a variable `net`, whose content depends on the communication model. The communication actions are two transition predicates `send` and `receive` which are true when the action is enabled. These actions take three parameters, the sender process, the destination process and the message. Their specification depends on the communication model. For instance, `NetworkFifo` specifies a communication model where the delivery order is globally first-in first-out: messages are delivered in the order they have been sent. Its realization is a queue and the two predicates are:

$$\begin{aligned} send(from, to, m) &\triangleq net' = Append(net, \langle from, to, m\rangle) \\ receive(from, to, m) &\triangleq net \neq \langle\rangle \wedge \langle from, to, m\rangle = Head(net) \wedge net' = Tail(net) \end{aligned}$$

3.4 Mechanized Verification

A specific BPMN diagram is described by instantiating the constants in `PWSDefs` (`Node`, `Edge`...) from the BPMN collaboration. This is automated using our `fbpmn` tool. Regarding the well-formedness of the BPMN diagram, the predicates from `PWSWellFormed` are *assumed* in the model. Before checking a model, The TLA$^+$ model checker checks these assumptions with the instantiated constants that describe the diagram, and reports an error if an assumption is violated. Otherwise, this proves that the diagram is well-formed.

The TLA$^+$ model checker, TLC, is an explicit-state model checker that checks both safety and liveness properties specified in LTL. This logic includes operators $\Box$ and $\Diamond$ that respectively denote that, in all executions, a property F must always hold ($\Box F$) or that it must hold at some instant in the future ($\Diamond F$). TLC builds and explores the full state space of the diagram to verify if the given properties are verified. These properties are generic properties related to any Business Process diagram, or specific properties for a given diagram. Some of the generic properties are safe collaboration, sound collaboration and message-relaxed sound collaboration [8]. A collaboration is safe if no sequence flow holds more than one token:

$$\Box(\forall e \in E^{SF}, m_e(e) \leq 1) \tag{1}$$

A collaboration is sound if all processes are sound and there are no undelivered messages. A process is sound if there are no token on its inside edges, and one token only on its end events.

$$\begin{aligned} SoundProc(p) \stackrel{def}{=} &\forall e \in R(p) \cap E^{SF}, m_e(e) = 0 \\ &\wedge \forall n \in R(p) \cap N, (m_n(n) = 0 \vee (m_n(n) = 1 \wedge n \in N^{EE})) \\ Soundness \stackrel{def}{=} &\Diamond(\forall p \in N^P, SoundProc(p) \wedge \forall e \in E^{MF}, m_e(e) = 0) \end{aligned} \tag{2}$$

A collaboration is message-relaxed sound (3) if it is sound when ignoring messages in transit, *i.e.*, when ignoring the Message Flow edges. This is the same as (2) without the second conjunction.

Other generic properties are available, such as the absence of undelivered message or the possible activation which states there does not exist a task node (Abstract Task, Send Task, Receive Task) that is never activated in any execution. From a business process point of view, it means that there are no tasks in the diagram that are never used. This is expressed as $\forall n \in N^T : \mathbf{EF}(m_n(n) \neq 0)$ (TLC can check for the invalidity of the negation of this CTL formula).

Last, the user can also define business model properties concerning a specific diagram. For instance, one can check that the marking of a given node is bounded by a constant (*i.e.*, $\Box(nodemarks[\text{"}ConfirmBooking\text{"}]) \leq 1$), or that the activation of one node necessarily leads to the activation of another node ($\Box(nodemarks[\text{"}BookTravel\text{"}] \neq 0 \Rightarrow \Diamond(nodemarks[\text{"}OfferCompleted\text{"}] \neq 0)$).

When the model checker finds that a property is invalid, it outputs a counter-example trace that we animate on the BPM graphical model to help the user understand it. As TLC uses a breadth-first algorithm, this trace is minimal for safety properties.

3.5 Experiments

Experiments were conducted on a laptop with a 2.1 GHz Intel Core i7 processor (quad core) with 8 GB of memory. Results are presented in Table 3. The first column is the reference of the example in our archive. The characteristics of a model are: number of participants, number of nodes (incl. gateways), number of flow edges (sequence or message flows), whether the model is well-balanced (for each gateway with n diverging branches we have a corresponding gateway with n converging branches) and whether it includes a loop. The communication model is asynchronous (bag), fifo-ordered between each couple of processes (fifo pair), globally fifo (fifo all), or synchronous-like (RSC). The results of the verification then follow. First, data on the resulting transition system are given: number of states, number of transitions, and depth (length of the longest sequence of transitions that the model checker had to explore). For each of the three correctness properties presented above, we indicate if the model satisfies it. Lastly, the accumulated time for the verification of the three properties is given. Our tool supports more verifications (see Table 4) and can be easily extended with new properties. We selected these three ones since they are more BPMN specific [8].

Table 3 presents the results for a selection from our repository [19] for a variety of gateways and activities. These illustrative examples include realistic business process models (001 and 002 two client-supplier models, 006 from Figure 2, 017 from [14], and 020 from [7]), and models dedicated to specific concerns: termination end events and sub processes (007–011 from [8]), inclusive gateways (003, 012, 013 and 018), exclusive and event-based gateways (015 and 016).

A first conclusion is that verification is rather fast: the verification of one property generally takes just a few seconds per model, the longest being for model 020 that takes up to 53s of accumulated time for the three properties (5s for the construction of the state space). Experiments also show the effect of the communication model on property satisfaction (models 1, 2, 6, 20), the use of

Table 3: Experimental Results.

ref.	Characteristics					Com.	LTS size			validity			total
	proc.	nodes (gw.)	SF/MF	B	L	model	states	trans.	depth	(1)	(2)	(3)	time
001	2	17 (2)	14/3	✓	×	bag	93	173	25	✓	✓	✓	5.17s
						fifo pair	85	161	21	✓	×	×	5.19s
						RSC	77	147	19	✓	×	×	5.09s
002	2	16 (2)	13/3	✓	×	bag	83	154	24	✓	✓	✓	4.93s
						fifo pair	75	142	20	✓	×	×	4.74s
						RSC	67	128	18	✓	×	×	4.48s
003	1	14 (6)	16/0	×	✓	none	41	59	15	✓	✓	✓	2.89s
006	2	20 (4)	18/5	×	✓	bag	470	966	43	×	×	✓	8.98s
						fifo all	522	932	40	×	×	×	8.58s
						RSC	247	420	38	×	×	×	6.58s
007	1	8 (2)	7/0	×	×	none	44	73	15	×	×	×	2.52s
008	1	11 (2)	9/0	×	×	none	48	77	19	×	✓	✓	2.60s
009	2	12 (2)	9/1	×	×	bag	170	395	19	×	×	×	4.86s
010	2	15 (2)	11/1	×	×	bag	186	423	23	×	×	✓	5.32s
011	2	15 (2)	11/1	×	×	bag	100	209	21	×	✓	✓	4.73s
012	1	15 (8)	17/0	✓	✓	none	71	137	15	✓	✓	✓	3.16s
013	1	17 (8)	21/0	✓	✓	none	407	1049	15	✓	✓	✓	5.93s
018	1	19 (8)	25/0	✓	✓	none	4631	15513	18	✓	✓	✓	28.46s
015	2	14 (2)	10/2	×	×	bag	68	117	11	✓	×	×	4.67s
016	2	14 (2)	10/2	×	×	bag	36	53	11	✓	✓	✓	4.25s
017	1	32 (12)	36/0	×	×	none	93	141	37	✓	✓	✓	4.03s
020	4	39 (6)	34/8	×	✓	bag	4648	14691	54	✓	✓	✓	53.05s
020						fifo all	2564	6872	54	✓	✓	✓	28.25s
020						RSC	1224	3271	54	✓	×	×	18.77s

TLA$^+$ fairness to avoid infinite loops (12, 13, 18, 20), and the use of terminate end events combined with model constraints to deal with unsafety (6).

4 Related Work

The formalization of the BPMN execution semantics, and on a wider scale the formal study of business processes, is a very active field of research. We here focus on recent work that provides tool support for the verification of collaboration diagrams and communication features in BPMN [9, 13, 20, 7, 8, 6, 15]. We add [10] due to its role as a seminal paper and [21, 22] as recent representatives for the formal model they use. Table 4 gives a synthetic presentation of a comparison between these proposals and ours.

Some works are based on transformations, while others provide a direct semantics. The former rely on an intermediary model, while the later have the benefit to provide a direct link between BPMN constructs and the verification formalism. Our work is in this line. Further, the choice of FOL lets one implement the semantics in different tools, *e.g.*, TLA$^+$ as here or SMT solvers.

As far as the BPMN coverage criteria is concerned, we can observe that we are among the approaches with a high coverage. To make verification tractable, we have abstracted from the data and the multi-instance constructs, that are often related to data. Works taking data into account in verification either require to bound domains or operate on the basis of configurations (a state and a substitution from variables to closed terms). The former is still subject to state-space explosion, while the later is closer to animation than to full-fledged verification, and makes it impossible to verify a model for any possible initial value of the data. A perspective of our work is to rely on symbolic techniques instead [18].

The work in [7] provides a very rich formal semantics with animation capacities but does not enable verification in the large. Most of the work, still, support the verification of BP correctness properties or, at least, all-purpose formal properties (reachability, deadlock). The use of FOL to define the semantics, and its implementation in TLA$^+$, made it possible for us to implement quite easily these correctness checks as temporal logic properties to be checked against the model.

5 Conclusion & Future Work

In this work we have proposed a formalization of a subset of BPMN execution semantics. It supports interaction and is parametric with reference to the properties of the communication between participants. We have seen in Table 3 that these properties can have an effect on the satisfaction of correctness properties by the collaboration model. Communication-parametric verification techniques for BPMN are therefore helpful when it comes to use this standard in contexts such as the IoT [3, 17] where communication may vary. Our proposal is equipped with open source tools that are freely available and automatically perform the transformations and verification steps. A direct perspective of this work is the integration of our proposal as a plug-in of a platform for business processes that goes beyond modeling, namely ProM [2].

An on-going work is to replace the network theories that specify the communication model with a more general and versatile solution based on a communication framework we have developed [5]. This framework allows a large variety of configurations for the communication model, including which ordering policies are to be applied per participant, per couples of communicating participants, using priorities, or bounds on the number of messages in transit.

Some BPMN features that play a role in full-fledged executable collaboration have been discarded, in a first step, when selecting a relevant BPMN subset. This is the case of the support for data and the multi-instances activity, that are supported for example in [7] to provide the business process designers with model animation. However, to keep verification tractable, we had to abstract from these features. Our main perspective goes in this direction, using techniques from symbolic execution and symbolic transition systems [18] while keeping a direct semantics, *i.e.*, relatable to the parts of the BPMN informal semantics.

References

1. Business Process Modeling Notation, http://www.omg.org/spec/BPMN/2.0.2/
2. ProM Framework, http://www.processmining.org/prom
3. Casati, F., et al.: Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks. In: Proc. of ICSE (2012)
4. Chevrou, F., Hurault, A., Quéinnec, P.: On the Diversity of Asynchronous Communication. Formal Aspects of Computing **28**(5), 847–879 (Sep 2016)
5. Chevrou, F., Hurault, A., Quéinnec, P.: A Modular Framework for Verifying Versatile Distributed Systems. Journal of Logical and Algebraic Methods in Programming (to appear)
6. Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F.: A Formal Approach to Modeling and Verification of Business Process Collaborations. Science of Computer Programming **166**, 35–70 (2018)
7. Corradini, F., Muzi, C., Re, B., Rossi, L., Tiezzi, F.: Animating Multiple Instances in BPMN Collaborations: From Formal Semantics to Tool Support. In: Proc. of BPM (2018)
8. Corradini, F., Muzi, C., Re, B., Tiezzi, F.: A Classification of BPMN Collaborations based on Safeness and Soundness Notions. In: Proc. of EXPRESS/SOS (2018)
9. Dechsupa, C., Vatanawood, W., Thongtak, A.: Transformation of the BPMN Design Model into a Colored Petri Net using the Partitioning Approach. IEEE Access **6**, 38421–38436 (2018)
10. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and Analysis of Business Process Models in BPMN. Information and Software technology **50**(12), 1281–1294 (2008)
11. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous Soundness Checking of Industrial Business Process Models. In: Proc. of BPM (2009)
12. Houhou, S., Baarir, S., Poizat, P., Quéinnec, P.: A First-Order Logic Semantics for Communication-Parametric BPMN Collaborations (Extended Version), available from: http://pardi.enseeiht.fr/BPM19
13. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal Verification of Complex Business Processes Based on High-level Petri nets. Information Sciences **385**, 39–54 (2017)
14. Krishna, A., Poizat, P., Salaün, G.: Checking Business Process Evolution. Science of Computer Programming **170**, 1–26 (2019)
15. Lam, V.S.: A Precise Execution Semantics for BPMN. International Journal of Computer Science **39**(1), 20–33 (2012)
16. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison Wesley (2002)
17. Meyer, S., Ruppen, A., Hilty, L.: The Things of the Internet of Things in BPMN. In: Proc. of CAiSE Workshops (2015)
18. Nguyen, H.N., Poizat, P., Zaïdi, F.: A Symbolic Framework for the Conformance Checking of Value-Passing Choreographies. In: Proc. of ICSOC (2012)
19. Poizat, P., Quéinnec, P.: Formal Tool for BPMN. https://github.com/pascalpoizat/fbpmn (2019)
20. Van Gorp, P., Dijkman, R.: A Visual Token-based Formalization of BPMN 2.0 based on in-place Transformations. Information and Software Technology **55**(2), 365–394 (2013)

21. Wong, P.Y., Gibbons, J.: Formalisations and Applications of BPMN. Science of Computer Programming **76**(8), 633–650 (2011)
22. Ye, J., Sun, S., Song, W., Wen, L.: Formal Semantics of BPMN Process Models using YAWL. In: Proc. of IITA (2008)

Table 4: Comparison of tool-supported approaches for the analysis of communication in BPMN.

approach		transformation						direct				
reference		[9]	[13]	[20]	[21]	[22]	[10]	[7]	[8]	[6]	[15]	**ours**
year		2018	2017	2013	2011	2008	2008	2018			2012	2019
formalism		CPN	ECATNets	In-Place Graph	CSP	YAWL Nets	RDP	LTS			LTL	FOL
tool		not avail.	not avail.	yes	yes	yes	yes	yes	–	yes	not avail.	yes
		CP4PBMN	BPMNChecker	GrGen	OWorkflow	BPMN2YAWL	transformer	MIDA	–	BProVe	prototype	fBPMN
supported elements	*AND* gateway	●	●	●	●	●	●	●	●	●	●	●
	XOR gateway	●	●	●	●	●	●	●	●	●	●	●
	OR gateway	●	–	●	●	●	●	●	●	●	●	●
	EB gateway	●	●	●	–	●	●	●	●	●	●	●
	ST	●	●	●	●	●	●	●	●	●	●	●
	RT	●	●	●	●	●	●	●	●	●	●	●
	MSE	●	●	●	–	●	●	●	●	●	●	●
	TMIE	●	–	●	–	–	●	●	●	●	●	●
	CMIE	●	–	●	–	–	●	●	●	●	●	●
	MEE	●	●	●	–	–	–	●	●	●	●	●
	TEE	–	–	●	●	–	–	●	●	●	●	●
	sub-processes	●	●	●	–	●	●	●	–	–	●	●
	data	●	●	●	–	–	–	●	–	–	–	–
	multi instance (pools)	–	–	●	–	–	–	●	–	–	–	–
	multi instance(activities)	●	●	●	●	●	●	●	–	–	–	–
verification	option to compl.	–	●	–	–	●	–	–	–	●	–	●
	proper compl.	–	–	–	–	●	●	–	–	●	–	●
	no dead activity	–	–	–	–	●	●	–	–	●	–	●
	safety	–	–	–	●	–	–	–	●	●	●	●
	collaboration soundness	–	–	–	–	–	–	–	●	–	–	●
	msg relaxed soundness	–	–	–	–	–	–	–	●	–	–	●
	undelivered msg	–	–	–	–	–	–	–	–	–	–	●
	general-purpose	●	●	●	●	●	●	–	●	●	●	●
communication models		–	–	–	–	–	–	–	–	–	–	●