An Operational Semantics of BPMN Collaboration*

Flavio Corradini, Andrea Polini, Barbara Re, and Francesco Tiezzi

School of Science and Technology, University of Camerino, Italy
name.surname@unicam.it

Abstract. In the last years we are observing a growing interest in formalising the execution semantics of business process modelling languages that, despite their lack of formal characterisation, are widely adopted in industry and academia. In this paper, we focus on the OMG standard BPMN 2.0. Specifically, we provide a direct formalisation of its operational semantics in terms of Labelled Transition Systems (LTS). This approach permits both to avoid possible miss-interpretations due to the usage of the natural language in the specification of the standard, and to overcome issues due to the mapping of BPMN to other formal languages, which are equipped with their own semantics. In addition, it paves the way for the use of consolidated formal reasoning techniques based on LTS (e.g., model checking). Our operational semantics is given for a relevant subset of BPMN elements focusing on the capability to model collaborations among organisations via message exchange. Moreover, one of its distinctive aspects is the suitability to model business processes with arbitrary topology. This allows designers to freely specify their processes according to the reality without the need of defining well-structured models. We illustrate our approach through a simple, yet realistic, running example about commercial transactions.

Keywords: Business Process Modelling, BPMN Collaboration, Operational Semantics

1 Introduction

Organisations, such as big companies or public administrations, nowadays operate in complex and volatile contexts, that ask for prompt reactions to emerging changes in order to maintain competitiveness and efficiency. To answer to such a need, in the last years a lot of effort has been put in the definition of modelling languages and tools permitting to represent and reason on different perspectives of such organisations. Among the others, Business Process (BP) modeling is certainly the activity that received the most attention, given its relevance in the reflection and definition of strategies for the alignement of introduced IT systems and business activities. A BP is described as "*a collection of related and structured activities undertaken by one or more organisations in order to pursue some particular goal. Within an organisation a BP results in the provisioning of services or in the production of goods for internal or external stakeholders. Moreover BPs are often interrelated since the execution of a BP often results*

* This research has been partially founded by EU project LearnPAd (GA:619583) and by the Project MIUR PRIN CINA (2010LHT4KM).

in the activation of related BPs within the same or other organisations" [1]. In deriving a BP model many different information and perspectives of an organisation can be captured [2]. Among the others we focus on: information related to the activities to be performed (function perspective), who should perform them (organisation perspective), when they should be performed and how they are organised in a flow (behaviour perspective). Many different languages and graphical notations have been proposed to represent BP models with differences both in the possibility to express aspects related to the perspectives, and in the level of formality used to define the elements composing the notation. BPMN 2.0[1], which has been standardised by OMG [3], is currently acquiring a clear predominance, among the various proposal, due to its intuitive graphical notation, the wide acceptance by industry and academia, and the support provided by a wide spectrum of modelling tools[2].

BPMN's success comes from its versatility and capability to represent BPs with different levels of detail and for different purposes. The notation acquired, at first, acceptance within business analysts and operators, who use it to design BP models. Successively, it has been more and more adopted by IT specialists to lead the development and settlement of IT systems supporting the execution of a BP model. Among the various characteristics of the notation, particularly interesting is the possibility to model a *collaboration* of different organisations exchanging messages and cooperating to reach a shared business goal. Collaboration diagrams are indeed the focus of our work since they contains enough information to assess the alignment of participants behavior, and the message flow specified to permit successful cooperations. If from the point of view of the notation the inter-organisation message exchange could seem a simple graphical element, its impact is absolutely relevant. When a modelling notation is used in a homogeneous context, such as a single organisation, the precise definition of the meaning of the various elements constituting the notation can be sometime avoided. Nevertheless, mutual understanding is possible thanks to the direct communications among the involved stakeholders, and from the emergence of established and accepted practices. This is not the case when two or more organisations are involved. In particular, in order to correctly collaborate, the involved organisations have to share the same understanding of communication mechanisms. Moreover, when a BP model includes the specification of collaborations among more organisations, it becomes fundamental that they can rely on a shared understanding of the model. In the last years, a relevant effort has been devoted by the research community to provide a formal semantics to the BPMN notation (we refer to Section 5 for an overview of major contributions on this side). Indeed, in defining the notation, OMG did not intend to provide a rigorous semantics for the various graphical elements; instead the meaning is given using natural language descriptions, permitting a wider adoption of the notation in different contexts. The use of formal tools to define the semantics of the various elements, and hence of a BP model, is relevant in order to enable automatic analysis activities that allow the designers to check if the BP satisfies desired properties or not. This aspect seems to be even more relevant when organisations get in contact with each other and need to analyse the impact of collaborative actions. Consider for instance the merging of two companies, in

[1] We use BPMN or BPMN 2.0 interchangeably to refer to version 2.0 of the notation.

[2] BPMN is currently supported by 75 tools (see `http://www.bpmn.org` for a detailed list).

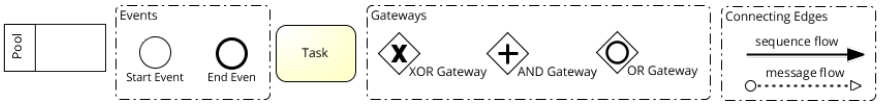


Fig. 1. Considered BPMN 2.0 Elements.

which there is not a common understanding of the models, and then the importance of analysis activities run to get a better understanding on the impact of the integration, and to discover flaws in the collaboration resulting from the possible integration.

In this paper, we intend to contribute to such a research effort aiming at providing a precise characterisation of BPMN elements with a special emphasis on communication within collaboration diagrams. This is mainly motivated by the need of achieving inter-organisation correctness, which is still a challenge [4]. More specifically, the contribution of the paper is a novel formalisation that provides an operational semantics to BPMN in the SOS style [5] by relying on the notion of Labeled Transition System (LTS). The major benefits of our semantics are as follows:

- it is a native semantics, rather than a mapping to other formalisms (equipped with their own semantics) like most of the proposals in the literature (see Section 5);
- it provides a compositional approach based on LTS, which paves the way for the use of consolidated analysis techniques and related software tools (see Section 6);
- it is suitable to model business processes with arbitrary topology, without imposing syntactical restrictions to the modeler, such as *well-structuredness* [6] (which, e.g., imposes gateways in a process to form single-entry-single-exit fragments) typically required by other proposals (see Section 5);
- besides core elements, such as tasks, gateways, etc., it takes into account collaborations and message exchange, which are overlooked by other formalisations.

The rest of the paper is organised as follows. Section 2 reports some background material on BPMN 2.0. Sections 3 and Section 4 introduce BPMN syntax and operational semantics we propose. Section 5 presents a detailed comparison of our approach with the related ones available in the literature. Finally, Section 6 closes the paper with some conclusions and opportunities for future work.

2 Background Notions on BPMN 2.0

The focus of this section is not a complete presentation of the standard, but a discussion of the main concepts of BPMN we use in the following. These concepts are briefly described below and reported in Figure 1. **Pools** are used to represent a participant or an organisation involved in the collaboration, and provide details on internal process specifications and related elements. Pools are drawn as rectangles. **Events** are used to represent something that can happen. An event can be a *Start Event*, representing the point in which the process starts, while an *End Event* is raised when the process terminates. Events are drawn as circles. **Tasks** are used to represent a specific work to perform within a process. Tasks are drawn as rectangles with rounded corners. **Gateways** are used to manage the flow of a process both for parallel activities and choices.

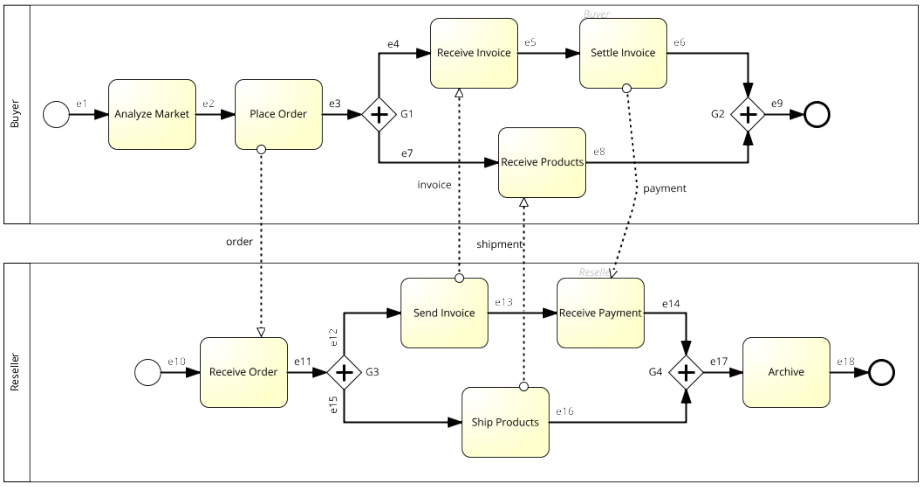


Fig. 2. Buyer-Reseller Example (source [7] p. 223).

Gateways are drawn as diamonds and act as either join nodes or split nodes. Different types of gateways are available, and we report here the most used ones. A *XOR gateway* gives the possibility to describe choices both in input (joining) and output (splitting); it is activated each time the gateway is reached and, when executed, it activates exactly one outgoing edge. An *AND gateway* enables a parallel flow execution: when used to split the sequence flow, all outgoing branches are activated simultaneously; when it joins parallel branches, it waits for all incoming branches to complete before triggering the outgoing flow. An *OR gateway* gives the possibility to select an arbitrary number of outgoing edges each time it is reached; all active incoming branches must complete before joining. Notably, even if XOR and OR splitting gateways may have guard conditions in their outgoing sequence flows. In this work we do not consider such possibility, as conditions have a significant role only when actual input values are taken into account, while our aim is to enable the verification of all possible flows of a process, and not only those triggered by specific input values. Finally, **Connecting Edges** are used to connect process elements in the same or different pools. Sequence flow is used to specify the internal flow of the process, thus ordering events, activities and gateways in the same pool, while message flow is a dashed connector used to visualise communication flows between organisations.

We introduce here a BPMN collaboration specification used throughout the paper as a running example.

Running Example (1/3). Figure 2 shows an example of BPMN process which combines the activities of a buyer organisation and a reseller one that have to interact in the market in order to complete a commercial transaction. After the buyer organisation analyses the market, it places its order by sending the *order* message to the reseller. Then, the buyer forks into two parallel paths by means of the AND gateway G1. The upper path receives the *invoice* from the reseller and settles it; in parallel the lower path receives the products from the reseller. Finally, the two flows of the buyer synchronise at the AND gateway G2 and the buyer stops its activities. This exchange of messages is supported

by the behaviour of the reseller that, after receiving the order, forks its behaviour into two parallel paths using the AND gateway G3. In the upper path, the reseller sends the *invoice* and receives the *payment*, while in the bottom one it performs the *shipment* of the ordered products. Finally, the flows of the reseller synchronise at the AND gateway G4 and the process of the reseller ends after the order is archived. □

It is worth noticing that we focus on the control flow and interacting aspects of business processes. This is mainly motivated by the need of keeping the semantics of the considered language rigorous but still manageable. Therefore, we intentionally left out other aspects, including timed events, data objects, sub-processing, error handling, and multiple instances. Instead, other aspects of BPMN can be easily rendered with our syntax, such as intermediate message events that can be reconducted to tasks with an incoming message flow. Anyway, we do not consider this restriction on the syntax as a major limitation, because we focus on the BPMN constructs most used in practice (indeed, even if the BPMN specification is quite wide, only less than 20% of its vocabulary is used regularly in designing BP models [8]).

3 BNF Syntax

The syntax of BPMN 2.0 is given in [3] by a metamodel in classical UML-style. In this section we provide an alternative syntax, in BNF-style, that is more suitable for defining a formal operational semantics.

The syntax is defined by grammar productions of the form $N ::= A_1 \mid \ldots \mid A_n$, where N is a non-terminal symbol and alternatives $A_1, \ldots, A_n$ are compositions of terminal and non-terminal symbols. In particular, in the grammar in Figure 3, the non-terminal symbols are C, P and G, representing *collaborations*, *processes* and *gateways*, respectively, while the terminal symbols are the typical graphical elements of a BPMN model, i.e. pools, events, tasks, gateways, and edges.

Intuitively, a BPMN collaboration model is rendered in our syntax as a collection of pools, where message edges can connect different pools. Each pool contains a process, defined as a collection of nodes, with incoming and/or outgoing sequence edges. Such nodes are events, tasks and (XOR/AND/OR) gateways. Notably, to obtain a compositional definition, each (message/sequence) edge is divided in two parts: the part outgoing from the source node and the part incoming into the target node. In fact, a term of the syntax can be straightforwardly obtained from a BPMN model by decomposing the collaboration in collection of pools, processes in collection of nodes, and edges in two parts.

We use the following disjoint sets of names: the set of *organisation* names (ranged over by o), the set of *message* names (ranged over by m), the set of *edge* names (ranged over by e), and the set of *task* names (ranged over by t). As a matter of notation, we use edges of the form $\_ \overset{m}{- - -} \_$ to denote edges of the form $\_ \overset{m}{- - \rightarrow}$ either incoming into or outgoing from pools/tasks.

We only consider specifications that are *well-defined*, in the sense that they comply with the following four syntactic constraints:

- Distinct pools (resp. tasks) have different pool (resp. task) names.

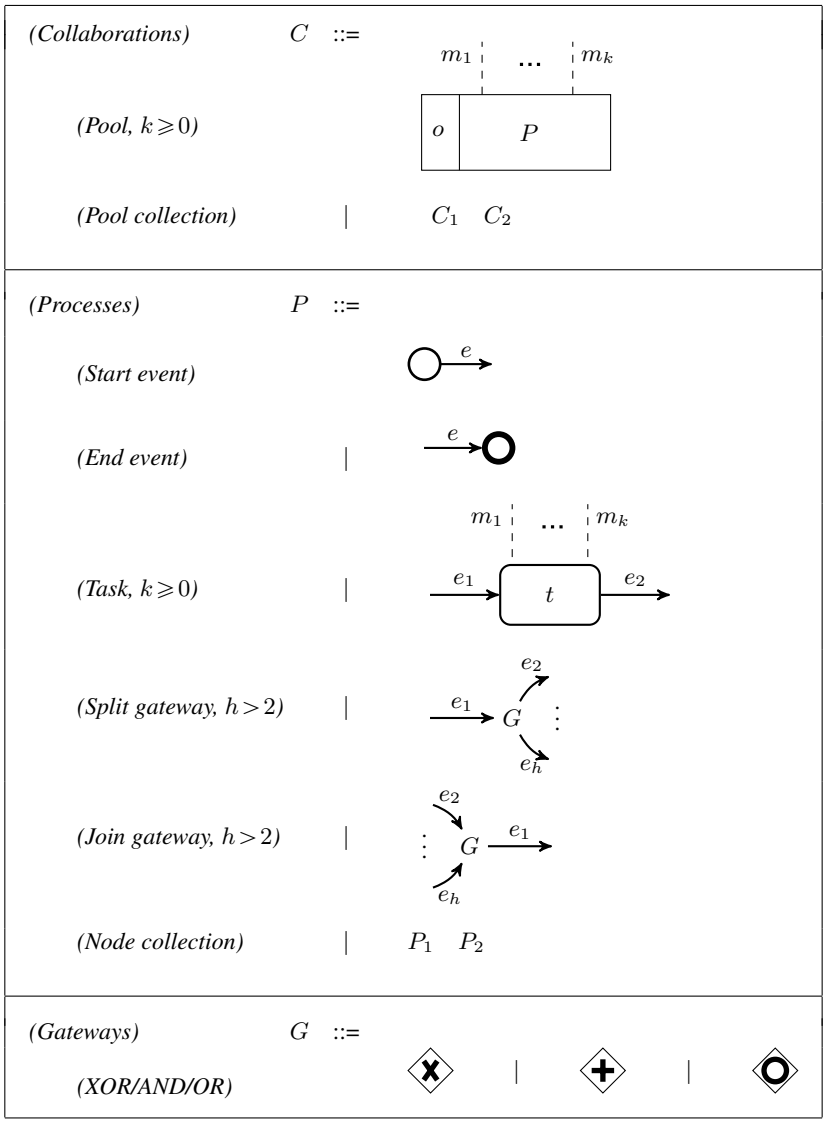


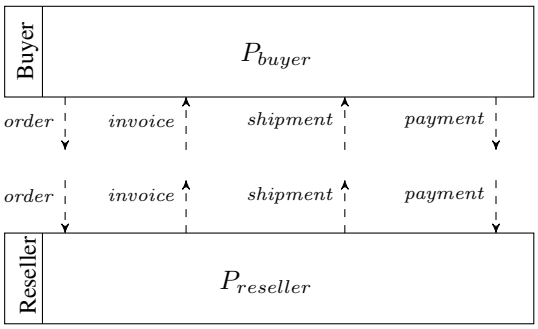
Fig. 3. BPMN Syntax

- In a collaboration, for each message edge labelled by m outgoing from a pool, there exists only one corresponding message edge labelled by m incoming into another pool, and vice versa.
- For each incoming (resp. outgoing) message edge labelled by m at pool level, there exists only one corresponding incoming (resp. outgoing) message edge labelled by m at the level of the process within the pool.
- In a process, for each sequence edge labelled by e outgoing from a node, there exists only one corresponding sequence edge labelled by e incoming into another node, and vice versa.

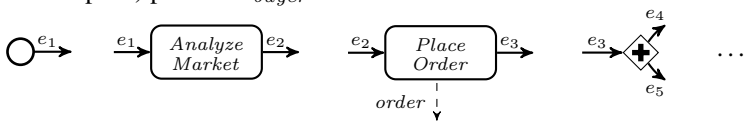
Well-definedness could be easily checked through a standard (and trivial) static analysis; more practically, the rationale is that each term of the language can be easily derived from a BPMN model whose only constraint is to have (pool, task, edge) unique names.

Notably, in this work we do not consider specifications using the OR join gateway, because formalising its semantics is a tricky task (see, e.g., [9] [10] [11]) that would make our formalisation much more complicated and, hence, out of focus.

Running Example (2/3). The BPMN model presented in Section 2 is expressed in our syntax as the following collaboration:



where (an excerpt of) process P_{buyer} is defined as follows:



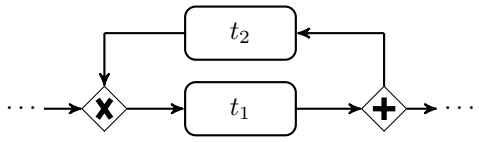
and process $P_{reseller}$ is defined in a similar way. □

4 Operational Semantics

We give the semantics of BPMN in terms of *marked collaborations*, i.e. collections of pools equipped with a marking. A *marking* is a distribution of tokens over pool message edges and process elements that indicate message arrivals and the process nodes that are active or not in a given step of the execution. This resembles the notions of token and marking in Petri Nets; this is not surprising as such formalism has strongly inspired the workflow constructs of BPMN. Similarly to the token-passing semantics in [12, 13], our tokens move along the syntax constructs, acting as sort of program counters.

For the sake of presentation, the operational semantics of BPMN is defined over an enriched syntax, w.r.t. the one given in Section 3, where pools' message edges are marked (i.e., labelled) by message tokens $\boxtimes$, while processes' edges, events and tasks are marked by workflow tokens $\bullet$. As a matter of notation, the presence of a number of message (resp. workflow) tokens in the same place is represented by means of one token of the form $\boxtimes n$ (resp. $\bullet\, n$), where $n \in \mathbb{N}_0$ is the token multiplicity. The initial marking of a collaboration assigns a single workflow token to the start events of the process of each pool in the collaboration. Notably, in this work we only consider business processes instantiated with single instances. In fact, dealing with multiple instances in

presence of message interactions would require to properly deliver each message to its appropriate instance; this would add complexity to our formal treatment, which we want to avoid in order to keep it as easy to understand as possible. On the other hand, the use of tokens with multiplicity is necessary also with single instances, e.g. due to the behaviour of the combined use of AND and XOR gateways as in the following piece of BPMN model:



Formally, the operational semantics of marked collaborations is defined in the SOS style by relying on the notion of Labeled Transition System (LTS). The labeled transition relation of the LTS defining the semantics of collaborations, at pool layer, is induced by the inference rules in Figure 4. We write $C \overset{l}{\longmapsto} C'$ to mean that "collaboration C can perform a transition labeled by l and become C' in doing so". Transition labels are generated by the following production rule:

$$(\mathit{Labels}) \qquad l \ ::= \ o : \alpha \quad | \quad o_1 \rightarrow o_2 : m$$

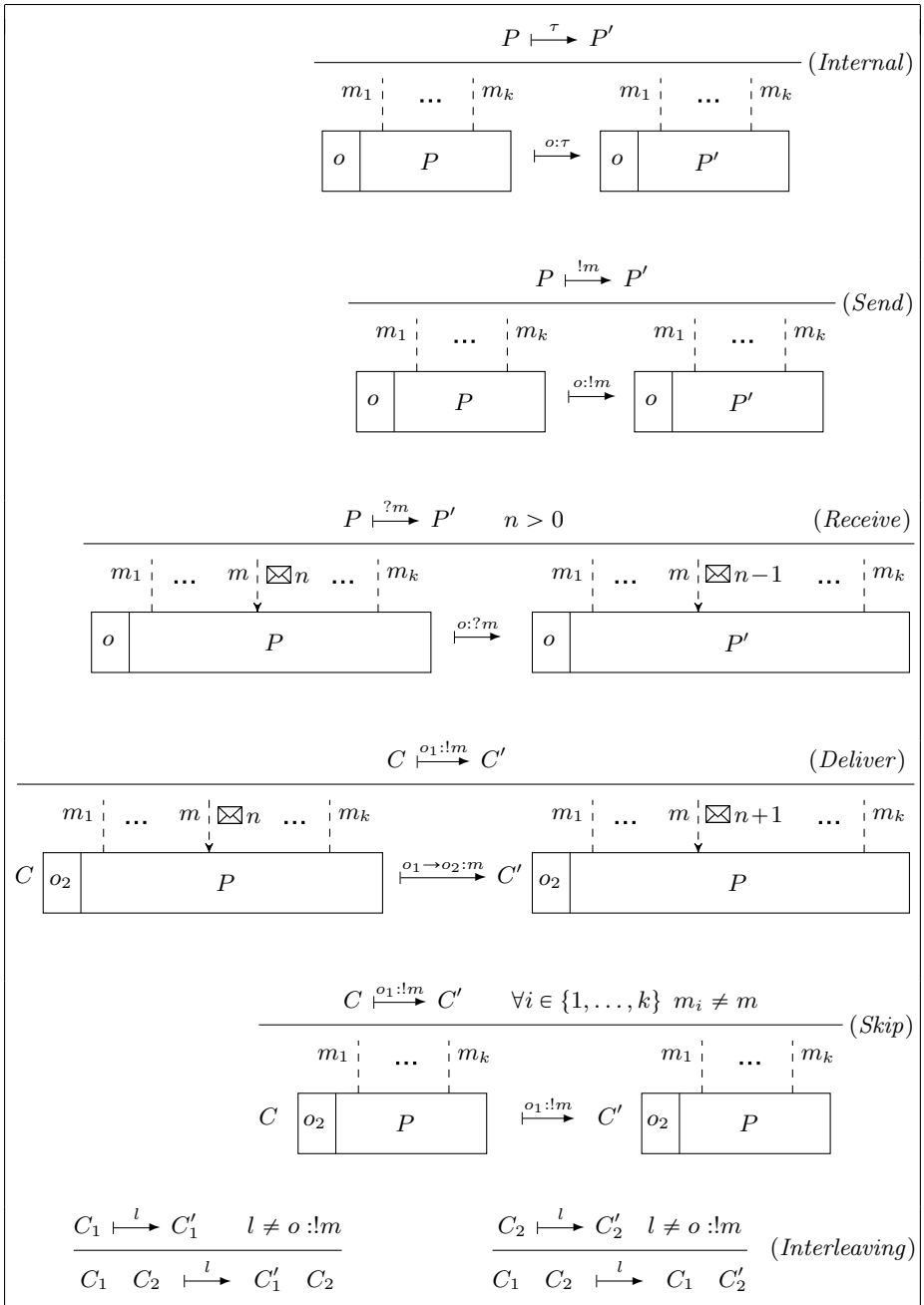
The meaning of labels is as follows: $o : \alpha$ denotes an action α peformed by the process instance of organisation o, while $o_1 \rightarrow o_2 : m$ denotes the exchange of a message m from organisation o_1 to o_2. The definition of the above relation relies on an auxiliary transition relation defining the semantics of process instances and induced by the inference rules in Figures 5, 6, and 7. We write $P \overset{\alpha}{\longmapsto} P'$ to mean that "process P can perform a transition labeled by α and become P' in doing so". The labels used by this auxiliary transition relation are generated by the following production rules:

$$(\mathit{Actions}) \qquad \alpha \ ::= \ \tau \quad | \quad !m \quad | \quad ?m$$

$$(\mathit{Internal\ actions}) \qquad \tau \ ::= \ \mathit{enabled}\ t \quad | \quad \mathit{completed}\ t \quad | \quad (-\tilde{e_1}, +\tilde{e_2})$$

where notation $\tilde{e}$ indicates a set of edges. The meaning of labels is as follows: τ denotes an action internal to the process, while $!m$ and $?m$ denote send and receive actions, respectively. The meaning of internal actions is as follows: $\mathit{enabled}\ t$ and $\mathit{completed}\ t$ denote the start and completion of the execution of task t, respectively; the pair $(-\tilde{e_1}, +\tilde{e_2})$ denotes movement of workflow tokens in the process graph, in particular one token is removed from each edge in $\tilde{e_1}$ and one is added to each edge in $\tilde{e_2}$ (whenever one of the two sets of edges is empty, its field is omitted from the pair).

We now briefly comment the rules in Figure 4. The first three rules allow a single pool, representing organisation o, to evolve according to the evolution of its enclosed process P. In particular, if P performs an internal action (rule *Internal*), a sending action (rule *Send*) or a receiving action (rule *Receive*), the pool performs the corresponding action at collaboration layer, i.e. the label is enriched with the name o of the organisation performing the action. Notably, rule *Receive* can be applied only if there is at least one ($n > 0$) message m queued in the corresponding message edge of the pool; of course,



Note: for the sake of presentation, in the rules above message tokens $\boxtimes n$ are omitted from message edges that are not involved in the considered transition; for example, in rule *Receive* the tokens of all message edges except for the edge labelled by m are omitted.

Fig. 4. BPMN Operational Semantics: Collaboration Layer

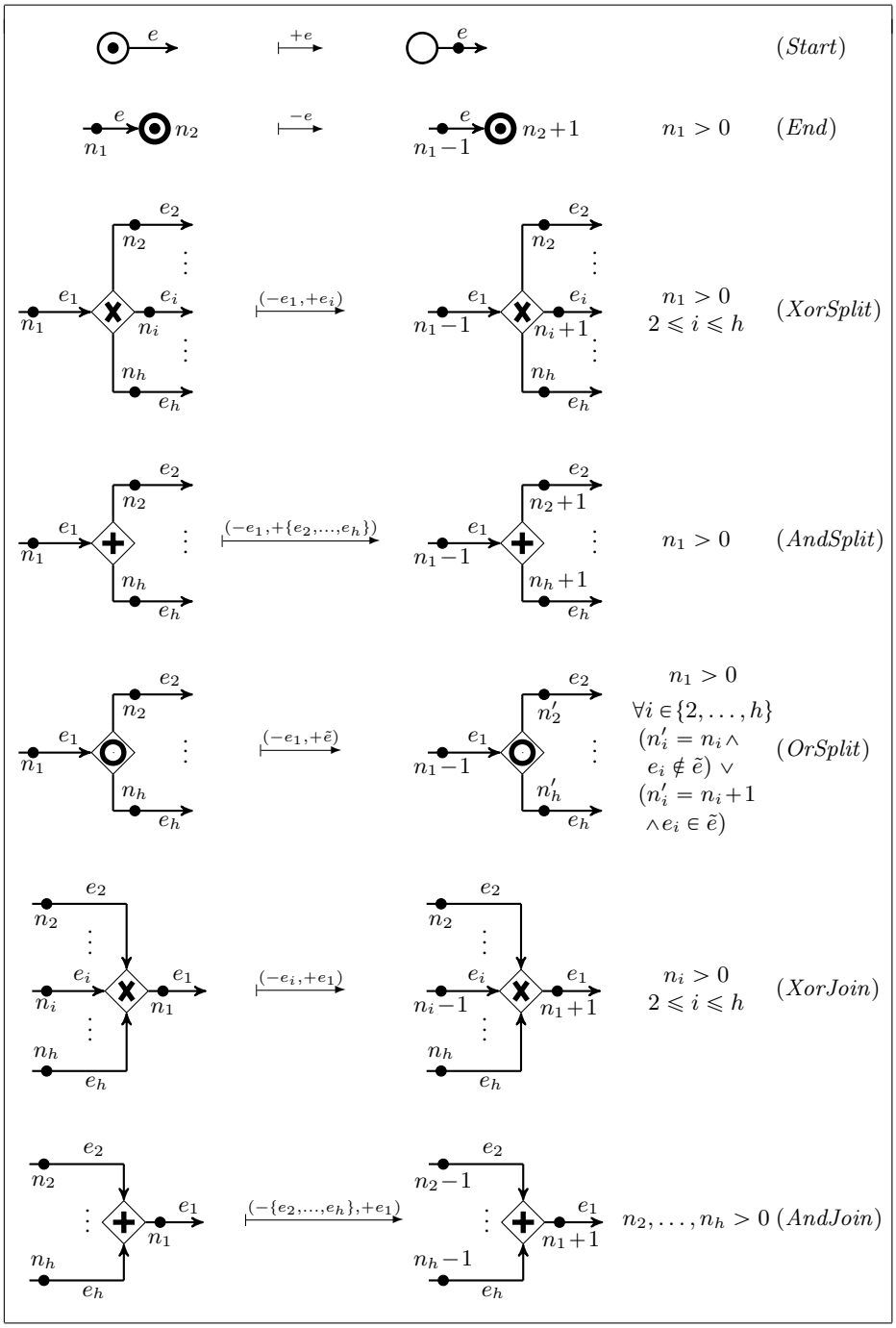


Fig. 5. BPMN Operational Semantics: Process Layer (Control Flow Constructs).

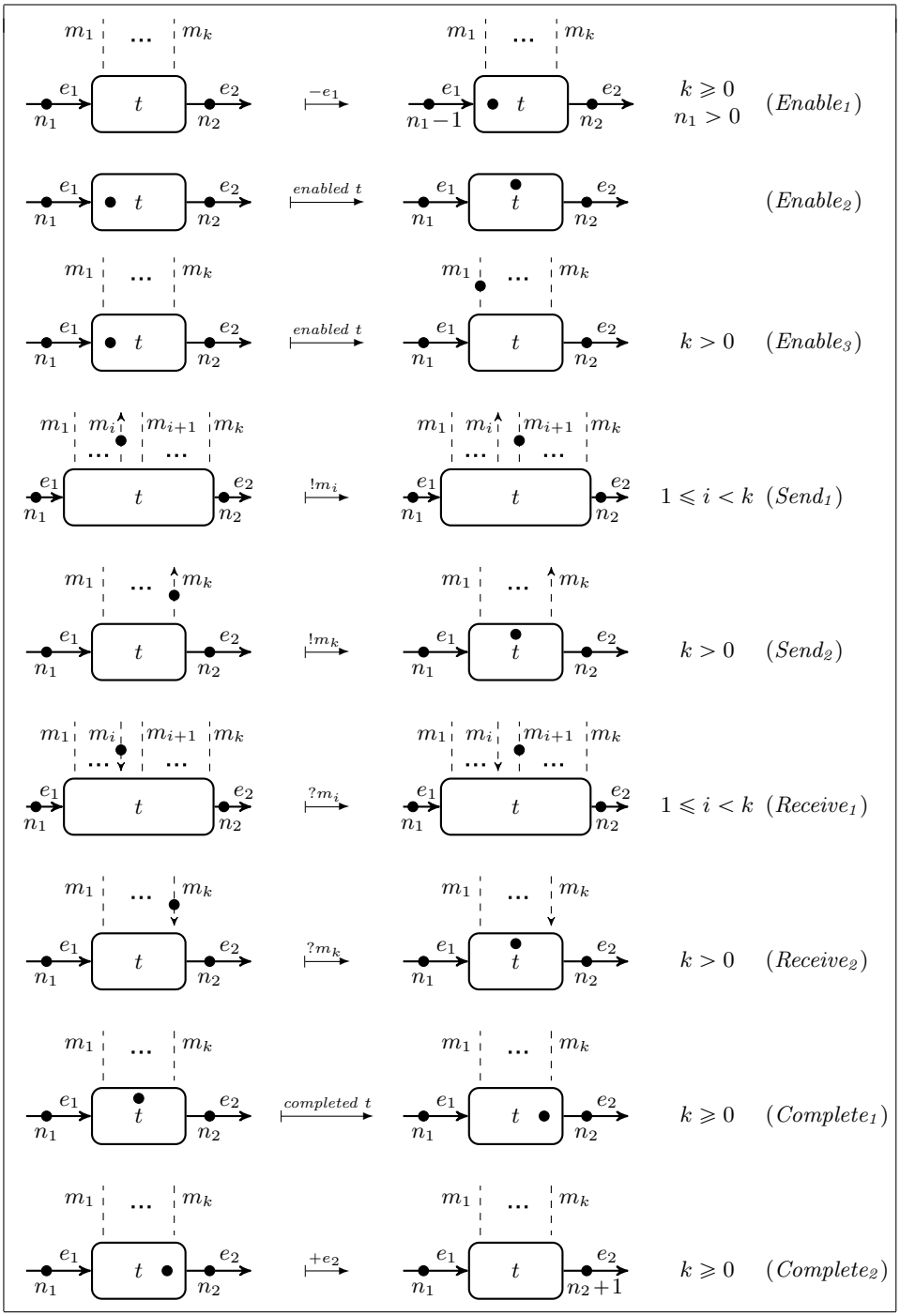


Fig. 6. BPMN Operational Semantics: Process Layer (Task Constructs).

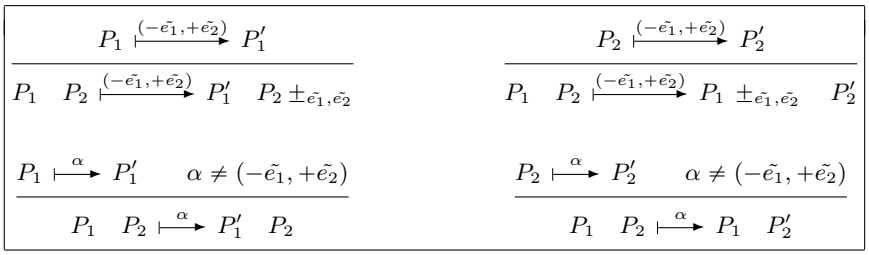


Fig. 7. BPMN Operational Semantics: Process Layer (Node Collection).

a message token is consumed by this transition. Instead, when an organisation o_1 indicates the willingness to send a message m (represented by a transition labelled by $o_1 :!m$), such message is properly delivered to the receiving organisation o_2 by applying rule *Deliver*. The resulting transition, labelled by $o_1 \rightarrow o_2 : m$, has the effect of increasing in the pool of o_2 the number of message tokens queued in the message edge labelled by m. If organisation o_2 does not have a message edge labelled by m, i.e. o_2 is not supposed to receive message m, no interaction between o_1 and o_2 takes place and label $o_1 :!m$ is propagated (rule *Skip*). It is worth noticing that, as prescribed by the BPMN 2.0 specification, inter-organisation communication is *asynchronous*: the sending action is not blocking, while the receiving one is blocking when there is no message token to consume. The two *Interleaving* rules permit to interleave the execution of actions performed by pools of the same collaboration, so that if a part of a larger collaboration evolves, the whole collaboration evolves accordingly. Interleaving is disallowed in case of a sending action, in order to force the use of rules *Deliver* and *Skip* for synchronising the sending pool with the receiving one. In fact, labels of the form $o_1 :!m$ are never exhibited by a well-defined collaboration (see Section 3), as they are just auxiliary labels used for properly, and compositionally, inferring transitions labelled by $o_1 \rightarrow o_2 : m$.

Rules in Figure 5 deal with control flow constructs, i.e. events and gateways. All these rules are axioms (i.e., they have no premises) producing transition labels of the form $(-\tilde{e_1}, +\tilde{e_2})$. This means that the effect of these rules is simply changing the marking of the process, i.e. moving workflow tokens among edges. For example, the effect of the rule *AndSplit* is to consume a token from the incoming edge e_1 of the AND gateway and to add a token to each outgoing edge e_i, with $2 \leqslant i \leqslant h$. The propagation of marking updates to other nodes of the process is dealt with by the interleaving rules in Figure 7 (see comments below).

Rules in Figure 6 are axioms devoted to the evolution of tasks. When a task is enabled (rule $Enable_1$), a token from its incoming edge is consumed and is placed on the left of the task name to indicate the starting status of the task. Notably, a task can be activated only when no token is placed inside the task rectangle or on its message edges; this means that parallel executions of the same task are not allowed. The fact that a task t is enabled is notified by applying either rule $Enable_2$ or $Enable_3$, depending on the presence of message edges. When a message edge is marked by a token, the corresponding sending or receiving action is performed; moreover the token is moved to the next edge (rules $Send_1$ or $Receive_1$) or on the top of the task name (rules $Send_2$ or $Receive_2$). Notice that the order of message edges is relevant for the execution: mes-

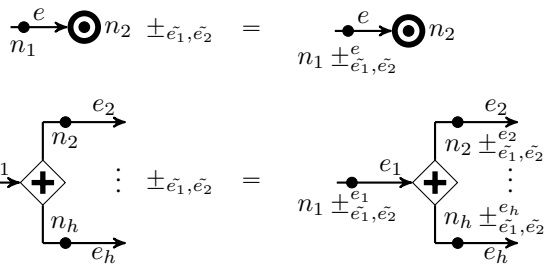
sages are processed from left to the right. This permits disambiguating the semantics of tasks in case of multiple message edges. Finally, when all messages are processed, the completion of the task execution is notified (rule $Complete_1$) and the number of tokens on the outgoing edge is increased by one (rule $Complete_2$).

The last group of rules, shown in Figure 7, deal with interleaving of process node evolutions. The first two rules are applied when the evolution involves a change in the marking of process edges, while the second two are applied in the other cases. In particular, the former rules relies on the marking updating function $P \pm_{\tilde{e_1},\tilde{e_2}}$, which returns a process obtained from P by unmarking (resp. marking) edges in $\tilde{e_1}$ (resp. $\tilde{e_2}$). Formally, this function is inductively defined on the structure of process P, by also relying on the following auxiliary function:

$$n \pm^{e}_{\tilde{e_1},\tilde{e_2}} = \begin{cases} n-1 & \text{if } e \in \tilde{e_1} \\ n+1 & \text{if } e \in \tilde{e_2} \\ n & \text{otherwise} \end{cases}$$

Notably, in the above definition we exploit the fact that, since self-loop are not admitted in a process, it holds $\tilde{e_1} \cap \tilde{e_2} = \varnothing$. In each base case of the inductive definition of the marking updating function, we simply apply the auxiliary function to the multiplicity of all tokens that mark an edge of the process node. We report below few significant cases of the definition (the others are similar):

$$(P_1 \quad P_2) \pm_{\tilde{e_1},\tilde{e_2}} = P_1 \pm_{\tilde{e_1},\tilde{e_2}} \quad P_2 \pm_{\tilde{e_1},\tilde{e_2}}$$



Running Example (3/3). We describe here the semantics of the BPMN model informally introduced in Section 2 and fomalised in Section 3. The initial state of the execution is represented by the collaboration in Figure 8(a), where the start events of the processes of the two organisations are marked by a workflow token each. Thus, the execution of both processes can start and, as a possible evolution, after few computational steps the status of the collaboration becomes the one shown in Figure 8(b). In such a configuration, according to the position of the two tokens, the buyer is performing the *Analize Market* task, while the reseller is already waiting for the order from the buyer. After other few steps, the collaboration status becomes the one in Figure 8(c), where the buyer has completed the *Analize Market* task and sent the *order* message to the reseller (as indicated by the message token ⊠ queued in the corresponding incoming message edge of the reseller's pool). Now, the reseller can consume the message and resume its computation. Finally, after further steps, the collaboration reaches the final configuration in Figure 8(d), where two workflow tokens mark the final events of the two processes. □

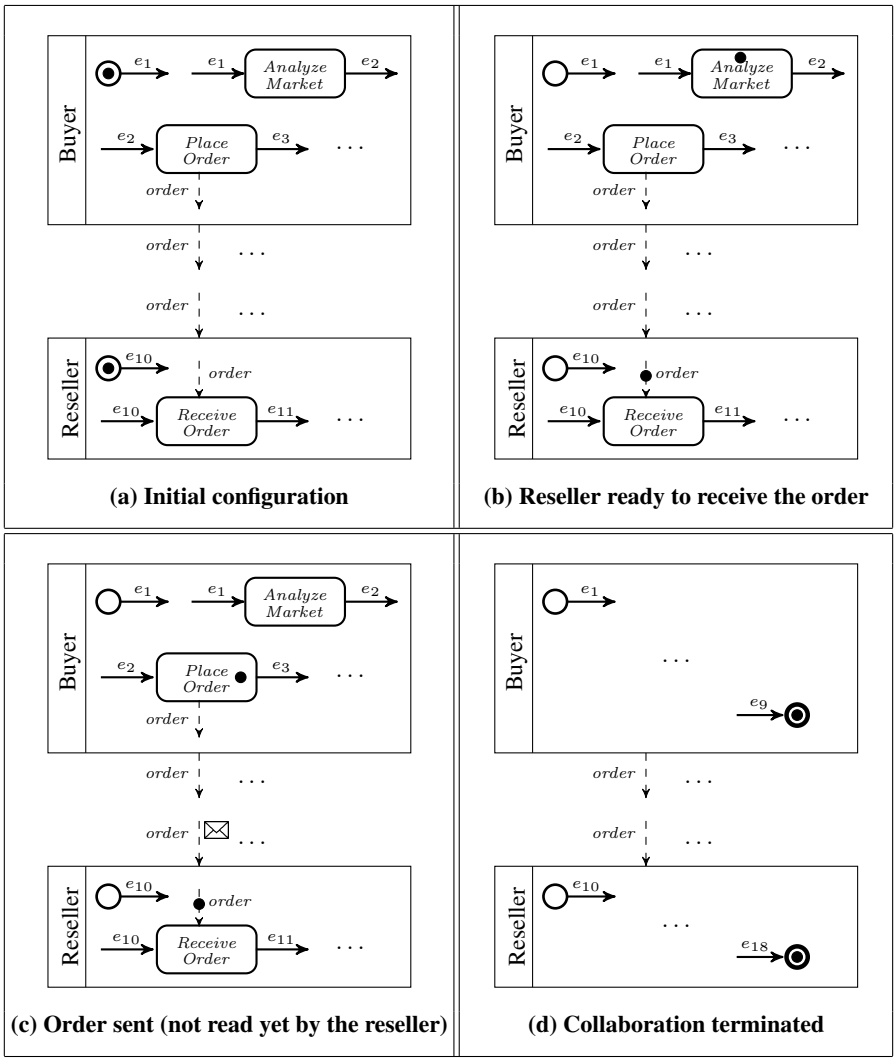


Fig. 8. Semantics of the running example: an excerpt

5 Related Work

Much effort has been devoted to the formalisation of BPMN. Here we refer to the most relevant attempts: we first consider the other direct formalisations available in the literature, then we discuss some mappings from BPMN to well-known formalisms.

With regard to direct formalisations, we refer to Van Gorp and Dijkman [14], Christiansen et. al [9], El-Saber and Boronat [15], and Borger and Thalheim [16]. Among them, our contribution was mainly inspired by the one presented in [14]. They propose a BPMN 2.0 formalisation based on in-place graph transformation rules; these rules are defined to be passed as input to the GrGen.NET tool, and are documented visually using

BPMN syntax. With respect to our work, the used formalisation techniques are different, since we provide an operational semantics in terms of LTS. This allows us to apply verification techniques based on transition labels, as e.g. model checking of properties expressed as formulae of action-based temporal logic. This gives us the possibility to be tool interdependent rather than be constrained to tools specific for graph transformation rules. Another interesting work is described in [9], where Christiansen et. al. propose a direct formalisation of the BPMN 2.0 Beta 1 Specification using algorithms based on incrementally updated data structures. The semantics is given for BPMNinc, that is a minimal subset of BPMN 2.0 containing just inclusive and exclusive gateways, start and end events, and sequence flows. This work differs from ours with respect to the formalisation method, as it proposes a token-based semantics à la Petri Nets, while we define an operational semantics with a compositional approach à la process calculi. Moreover, the work in [9] also lacks to take into account BPMN organisational aspects and the flow of messages, whose treatment is a main contribution of our work. El-Saber and Boronat proposed in [15] a formal characterisation of well-formed BPMN processes in terms of rewriting logic, using Maude as supporting tool. This formalisation refers to a subset of the BPMN specification considering elements that are used regularly, such as flow nodes, data elements, connecting flow elements, artefacts, and swimlanes. Interesting it is also the mechanism given to represent and evaluate guard conditions in decision gateways. Differently from the other direct formalisations, this approach can be only applied to well-structured processes. Concerning the well-structuredness requirement, we are aware that enforcing such restriction may have benefits, among which we refer to the importance of structuredness as a guideline to avoid errors in business process modelling [17]. But we are also aware that this requirement may result in a language more complex to use and less expressive [18]. We therefore consider the arbitrary topology as a benefit, because we assume that designers should be free to model the process according to the reality they feel without needing to define well-structured models. In addition, it should be considered that not all process models with an arbitrary topology can be transformed into equivalent well-structured processes [19, 20]. Moreover, the work in [15] has another drawback, concerning BPMN organisational aspects and messages flow. In particular, even if it is stated that messages are included in the formalisation, their formal treatment is not explained in the paper.

The most common formalisations of BPMN are given via mappings to various formalisms, such as Petri Nets [21–24, 6, 25], YAWL [26, 27] and process calculi [28–34]. This kind of formalisations suffers the typical problems introduced by a mapping. In fact, in these cases the semantics of BPMN is not given in terms of features and constructs of the language, but in terms of low-level details of their encodings. This makes the verification of BPMN models less effective, because the verification results refer to the low-level implementation of the models and may be difficult to be interpreted at BPMN level. Moreover, no formal proof of the correctness of these encodings with respect to a native semantics of BPMN is provided.

Regarding the mapping from BPMN to Petri Nets, the one proposed by Dijkman et. al. in [6] is probably the most relevant contribution. It enables the use of standard tools for process analysis, such as soundness of BPMN models. However, differently from our approach, even if the mapping deals with messages, it does not properly consider

multiple organisation scenarios, and does not provide information to the analysis phase regarding who are the participants involved in the exchange of messages.

Other relevant mappings are those from BPMN to YAWL, a language with a strictly defined execution semantics inspired by Petri Nets. Among the proposed mappings, we would like to mention the ones by Ye and Song [26] and Dumas et. al. [27]. The former is defined under the well-formedness assumption, which instead we do not rely on. Moreover, although messages are taken into account in the mapping, pools and lanes are not considered; thus it is not possible to identify who is the sender and who is the receiver in the communication. This results in the lack of capability to introduce verification at message level considering the involved organisations. The latter mapping, instead, formalises a very small portion of BPMN elements. In particular, limitations about pools and messages are similar to the previous approach: pools are treated as separate business processes, while messages flow is not covered by the mapping.

Process calculi has been also considered as means for formalising BPMN. Among the others, Wong and Gibbons presented in [29] a translation from a subset of BPMN process diagrams, under the assumption of well-formedness, to a CSP-like language based on Z notation. This enables the introduction of formal verification to check properties like consistency and compatibility. Even if messages have been omitted in the formalisation presented in [29], their treatment is discussed in [28]. Messages are also considered by Arbab et. al. in [30], where the main BPMN modeling primitives are represented by means of the coordination language Reo. Differently from the other mappings, this one considers a significantly larger set of BPMN elements. Prandi et. al., instead, defined in [31] a semantics in term of a mapping from BPMN to the process calculus COWS, which has been specifically devised for modelling service-oriented systems. Last but not least, also π-calculus was taken as target language of mapping by Hutchison et. al. [32] and Puhlmann [33]. Even if our proposal differs from the above ones, as it is a direct semantics rather than a mapping, it has drawn inspiration from those based on process calculi for the use of a compositional approach in the SOS style.

6 Concluding Remarks

The lack of a shared, well-established, comprehensive formal semantics for BPMN was the main driver of our work. This is also a critical point of the specification considering the wide adoption of the language both from the industry and research community. In this paper, we present an operational semantics in terms of LTS. We focus on the collaboration capability supported by message exchange. The proposed semantics enables designers to freely specify their processes with an arbitrary topology supporting the adherence to the standard, without the requirement of defining well-structured models.

The proposed formalisation allows one to verify properties on the model using consolidated formal reasoning techniques based on LTS. For instance, by expressing such properties by means of temporal logic, we can check, e.g., if *after* the enabling of a given task it can be *eventually* completed or not. More in general, we can verify, e.g., if *for all* possible executions all processes involved in a collaboration successfully terminates. This is quite relevant also with reference to the message exchange as, although communication is asynchronous, message receiving is blocking. We intend to investi-

gate verification of such kind of properties in the near future. We plan to achieve this by implementing our semantics in Maude[3] that allows to render operational rules of the semantics in terms of rewriting rules. This enables the (automatic or interactive) exploration of the evolutions of BPMN models, and it permits to exploit the rich analysis tool set provided by Maude. Even if we consider the use of Maude the most promising approach for our purposes, we plan to also investigate other approaches, such as [35, 36]. Moreover, we intend to develop a tool chain integrating the verification environment with a BPMN modelling environment, such as Eclipse BPMN Modeller[4]. This will offer the possibility of going back and forth between the modelling environment and the verification one, by e.g. graphically visualising on the BPMN model the feedbacks of the verification.

We also aim at extending our formalisation to model more BPMN elements, such as data objects, sub-processing, and error handling. In particular, we intend to focus on tricky issues concerning multiple instances of the same process and OR join gateway. Last but not least, we plan to prove some consistency properties of our operational semantics ensuring, e.g., that some syntactic constraints are preserved along the evolution of marked collaborations.

References

1. Lindsay, A., Downs, D., Lunn, K.: Business processes - attempts to find a definition. Information and Software Technology **45**(15) (2003) 1015–1019
2. Reichert, M., Weber, B.: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer (2012)
3. OMG: Business Process Model and Notation (BPMN v2.0), Normative document (Jan 2011)
4. Breu, R., Dustdar, S., Eder, J., Huemer, C., Kappel, G., Köpke, J., Langer, P., Mangler, J., Mendling, J., Neumann, G., Rinderle-Ma, S., Schulte, S., Sobernig, S., Weber, B.: Towards Living Inter-organizational Processes. In: CBI, IEEE (2013) 363–366
5. Plotkin, G.: A structural approach to operational semantics. J. Log. Algebr. Program. **60-61** (2004) 17–139
6. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology **50**(12) (2008) 1281–1294
7. Weske, M.: Business Process Management. Springer (2012)
8. Muehlen, M.z., Recker, J.: How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: CAISE. Volume 5074 of LNCS. Springer (2008) 465–479
9. Christiansen, D.R., Carbone, M., Hildebrandt, T.T.: Formal semantics and implementation of BPMN 2.0 inclusive gateways. In: WSFM. (2011) 146–160
10. Gfeller, B., Völzer, H., Wilmsmann, G.: Faster or-join enactment for bpmn 2.0. In: Business Process Model and Notation. Springer (2011) 31–43
11. Dumas, M., Grosskopf, A., Hettel, T., Wynn, M.: Semantics of standard process models with or-joins. In: On the Move to Meaningful Internet Systems. Springer (2007) 41–58
12. Sinot, F.: Call-by-Name and Call-by-Value as Token-Passing Interaction Nets. In: TLCA. Volume 3461 of LNCS., Springer (2005) 386–400

[3] `http://maude.cs.illinois.edu/`

[4] `http://www.eclipse.org/bpmn2-modeler/`

13. Kirchner, F., Sinot, F.: Rule-Based Operational Semantics for an Imperative Language. Electr. Notes Theor. Comput. Sci. **174**(1) (2007) 35–47
14. 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) (2013) 365–394
15. El-Saber, N., Boronat, A.: BPMN Formalization and Verification using Maude. In: BM-FA, ACM Press (2014) 1–12
16. Börger, E., Thalheim, B.: A Method for Verifiable and Validatable Business Process Modeling. In: Advances in Software Engineering. Volume 5316. Springer (2008) 59–115
17. Laue, R., Mendling, J.: The Impact of Structuredness on Error Probability of Process Models. In: Information Systems and e-Business Technologies. Volume 5. Springer Berlin Heidelberg, Berlin, Heidelberg (2008) 585–590
18. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J.: On Structured Workflow Modelling. In: Advanced Information Systems Engineering. Volume 1789. Springer Berlin Heidelberg, Berlin, Heidelberg (2000) 431–445
19. Polyvyanyy, A., Garca-Bauelos, L., Dumas, M.: Structuring acyclic process models. Information Systems **37**(6) (2012) 518–538
20. Polyvyanyy, A., Garcia-Banuelos, L., Fahland, D., Weske, M.: Maximal Structuring of Acyclic Process Models. The Computer Journal **57**(1) (January 2014) 12–35
21. Huai, W., Liu, X., Sun, H.: Towards Trustworthy Composite Service Through Business Process Model Verification. In: UIC/ATC, IEEE (2010) 422–427
22. Koniewski, R., Dzielinski, A., Amborski, K.: Use of Petri Nets and Business Processes Management Notation in Modelling and Simulation of Multimodal Logistics Chains. In: ECMS. (2006) 99–102
23. Ramadan, M., Elmongui, H.G., Hassan, R.: BPMN formalisation using coloured petri nets. In: SEA. (2011)
24. Awad, A., Decker, G., Lohmann, N.: Diagnosing and Repairing Data Anomalies in Process Models. In: Business Process Management Workshops. Number 43 in LNBIP, Springer (2010) 5–16
25. Corradini, F., Polini, A., Re, B.: Inter-organizational business process verification in public administration. Business Process Management Journal **21**(5) (2015) 1040–1065
26. Ye, J., Song, W.: Transformation of BPMN diagrams to YAWL nets. J. of Softw. **5**(4) (2010)
27. Decker, G., Dijkman, R., Dumas, M., Garca-Bauelos, L.: Transforming BPMN Diagrams into YAWL Nets. In: BPM. Volume 5240 of LNCS. Springer (2008) 386–389
28. Wong, P.Y., Gibbons, J.: Formalisations and applications of BPMN. Science of Computer Programming **76**(8) (2011) 633–650
29. Wong, P.Y.H., Gibbons, J.: A Process Semantics for BPMN. In: Formal Methods and Software Engineering. Volume 5256 of LNCS. Springer (2008) 355–374
30. Arbab, F., Kokash, N., Meng, S.: Towards Using Reo for Compliance-Aware Business Process Modeling. In: ISOLA. Volume 17 of CCIS., Springer (2008) 108–123
31. Prandi, D., Quaglia, P., Zannone, N.: Formal Analysis of BPMN Via a Translation into COWS. In: COORDINATION. Volume 5052 of LNCS., Springer (2008) 249–263
32. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In: BPM. Volume 4102 of LNCS. Springer (2006) 145–160
33. Puhlmann, F.: Soundness Verification of Business Processes Specified in the Pi-Calculus. In: OTM. Volume 4803 of LNCS. Springer (2007) 6–23
34. Corradini, F., Polini, A., Polzonetti, A., Re, B.: Business processes verification for e-government service delivery. Information Systems Management **27**(4) (2010) 293–308
35. Lucanu, D., Serbanuta, T., Rosu, G.: K Framework Distilled. In: WRLA. Volume 7571 of LNCS., Springer (2012) 31–53
36. Rosu, G., Stefanescu, A.: Matching logic: a new program verification approach. In: ICSE, ACM (2011) 868–871