The Journal of Systems and Software 166 (2020) 110594

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Correctness checking for BPMN collaborations with sub-processes

Flavio Corradini, Andrea Morichetta*, Andrea Polini, Barbara Re, Lorenzo Rossi, Francesco Tiezzi

Computer Science Division, School of Sciences and Technology, University of Camerino Italy

ARTICLE INFO

Article history:
Received 6 August 2019
Revised 9 January 2020
Accepted 30 March 2020
Available online 6 April 2020

Keywords:
BPMN 2.0
Collaborations
Sub-Processes
Message flow
Formal verification

ABSTRACT

BPMN collaboration models are commonly used to describe the behaviour and interactions of processes in an inter-organisational context. An important role in this kind of models is played both by the message flow, and by sub-processes. The interplay between these features of BPMN models can conceal subtle or unexpected effects, which makes the design activity error-prone, thus leading to the possible inclusion of incorrect behaviour. In this paper, we face this problem by providing a framework for checking the correctness of BPMN models. In particular we are interested on collaboration models that include message exchange and/or sub-processes, and with a special focus on properties well-established in the business process domain, namely safeness and soundness. To enable such a verification, we have (i) defined an operational semantics for BPMN collaborations, (ii) formalised safeness and soundness properties, and a new relaxed version of soundness for detecting situations where asynchronous messages are not handled correctly by the receiver, (iii) applied the related checks on state-space representations (i.e., labelled transition systems) of collaborations, and (iv) implemented the overall formal framework that has been also integrated in the Camunda modelling environment. The resulting verification framework and tool, named $\mathcal{S}^3$, have been validated in relation to its effectiveness, efficiency and usability, both by using models available on a publicly accessible repository, and by carrying out experiments with a group of designers.

1. Introduction

Designing inter-organisational information systems generally require to provide descriptions of the system at different levels of abstraction. On the one hand, the model designer can better manage the complexity he/she has to handle going from abstract descriptions of the system towards more detailed ones, so to focus from time to time on specific aspects of the system, or parts of it. On the other hand, going from a detailed description to an abstract one permits to have models that are less crowded and more understandable to the interested readers, thus keeping the information flow in line with general comprehension capabilities (Corradini et al., 2018). A further relevant aspect of inter-organisational information systems concerns the capability of the involved participants to cooperate by exchanging messages.

Nowadays, the most widely used notation to model inter-organisational information systems is the BPMN 2.0 standard[1] (OMG, 2011). The notation offers *collaboration* diagrams that provide a way to represent both the *message exchange* among different participants, and details on the *process* of every single participant. Besides, within a process, the *sub-process* element can be used to represent a compound activity that can be expanded or collapsed at designer convenience to regulate the abstraction level of the model. However, the combined usage of messages and sub-processes in a collaboration model can lead to intricate and cumbersome behaviours, which may hide undesired situations not promptly identifiable by the designer. In the business process modelling domain such undesirable behaviours are typically related to the violation of general correctness properties expressly defined in workflow net, such as *safeness* (van der Aalst, 2000) and *soundness* (Van der Aalst, 1995). Soundness requires that a process can not run into a deadlock or in an undesired state where the control flow leads to execute the same part doubly (lack of synchronisation). Safeness requires not to have multiple tokens in a place rep-

* Corresponding author.
E-mail addresses: flavio.corradini@unicam.it (F. Corradini), andrea.morichetta@unicam.it (A. Morichetta), andrea.polini@unicam.it (A. Polini), barbara.re@unicam.it (B. Re), lorenzo.rossi@unicam.it (L. Rossi), francesco.tiezzi@unicam.it (F. Tiezzi).

https://doi.org/10.1016/j.jss.2020.110594
0164-1212/

[1] BPMN (Business Process Model and Notation) is an OMG standard for modelling business process (OMG, 2011).

resenting a condition. Unfortunately, although much formalism has been used in BPMN processes verification, like Petri Nets (Murata, 1989; Lohmann et al., 2009, Workflow Nets (van der Aalst, 2000), and Elementary Nets (Rozenberg and Engelfriet, 1998), there is a lack of support to check such correctness properties of BPMN collaboration models due to the difficulty to distinguish messages from control flow in the model. Therefore studying these properties directly on BPMN does not leave any room for ambiguity, and increases the potential for formal reasoning on BPMN. Furthermore, the majority of correctness tool in the literature consider sub-process as a normal process; merging the content in a global flatted process is incorrect according to the standard (OMG, 2011). This impact, even more, the correctness check when message exchanges and sub-process are combined since possible messages can be lost during the execution.

In this paper, our intention is to introduce a unique formal framework to allow BPMN designers, both novice and experienced, to achieve a better understanding of their models, and relative properties. This results in a systematic methodological approach to improve the design of BPMN collaborations. We face this crucial issue by providing **a framework, based on formal methods, enabling the verification of safeness and soundness properties for collaboration diagrams, taking into account distinctive characteristics introduced by message exchanges and sub-process elements**. More specifically, the major contributions of this paper are as follows discussed.

We have defined a **structural operational semantic** for rigorously characterising the semantics of the considered class of BPMN models considering as first-class citizens BPMN specificities, reasoning on collaboration, process and sub-process levels and asynchronous communication. The proposed operational semantics associates to each collaboration diagram a formal model in the form of a Labelled Transition System (LTS). In our framework, we do not impose any syntactical restriction on the usage of the modelling notation, such as *well-structuredness* (which, roughly, imposes gateways in a process to form single-entry-single-exit fragments). Despite the fact that we are aware of the benefits of structuredness in process modelling (Laue and Mendling, 2008; Dumas et al., 2012), we aim at removing such restriction by considering process models with an arbitrary topology, in order to give more flexibility and freedom to designers. Indeed, designers often do not adopt a well-structured approach (Polyvyanyy and Bussler, 2013), because the modelling activity results to be less complex (Kiepuszewski et al., 2013) and the models more expressive (it is well known that not all process models with an arbitrary topology can be transformed into equivalent well-structured processes (Polyvyanyy et al., 2012; 2014)). In addition, well-structuredness per se does not guarantee that models are correct (e.g., sound Dehnert and Zimmermann, 2005), while it is mainly a way contributing at solving some modelling issues (van der Aalst, 2000).

We have **introduced the notion of safeness, soundness and a novel relaxed variant of the soundness property**, which is specially devised for dealing with the exchange of messages in collaboration diagrams. Safeness of a BPMN collaboration only refers to tokens on the sequence edges, while in the Petri Nets translation (e.g., Dijkman et al., 2008) refers to tokens both on message and sequence edges. This is due to an inaccurate mapping that considers a message as a (standard) sequence edge token in a place. Hence, a safe BPMN collaboration where the same message is sent more than once (e.g., via a loop), it is erroneously considered unsafe by relying on the Petri Nets notion (i.e., 1-boundedness), because pending messages are rendered as a place with more than one token. Therefore, the notion of safeness defined for Petri Nets cannot be safely applied as it is to collaboration models.

Similarly, regarding the soundness property, we can consider different notions of soundness according to the requirements we impose on message queues (e.g., ignoring or not the pending messages). Again, due to lack of distinction between message and sequence edges, this fine-grained reasoning is precluded using the current translations from BPMN to Petri Nets. For these reasons **we have introduced the new message relaxed sound notion**, that does not require that the receiver handles each asynchronously exchanged message. Checking this property in combination with the usual one permits to spot those *warning* situations in which the control flow execution completes but some messages are pending. These situations, indeed, do not necessarily correspond to incorrect behaviours. For example, in case of a race condition from messages caused by an event-based gateway, only one message among the waited ones is consumed, while the others remain enqueued. Instead, the case where decisions have enabled a control flow that skips the reception of a message may be either undesirable or acceptable, depending on the specific application context. In such a case, the model designer is in charge of checking the nature and relevance of the pending message. The presented notions are directly defined on the BPMN collaborations, and there is a clear difference compared to the one defined on Petri Nets and applied to the Petri Nets resulting from the translation of BPMN collaborations, e.g. via the mappings in Dijkman et al. (2008) and in Matthias and Mathias (2016).

We have **implemented in Java the operational semantics and the correctness checking techniques**. It results in efficient checking techniques on LTSs for the considered properties, and proved that these checks correspond to the formal definitions of the properties. In particular, we provide a *verification tool*, called $\mathcal{S}^3$, accessible as a RESTful service, as a web-application based on the Camunda modelling environment (https://bpmn.io/modeler/), and as a Java stand-alone application. Last but not least, we have **extensively validated the proposed verification framework and the $\mathcal{S}^3$tool**. Firstly, we have checked if safeness, soundness and message-relaxed soundness are correctly handled by model designers, or if, instead, they release models violating such properties. We did such investigation using $\mathcal{S}^3$ on BPMN models publicly available on a freely accessible repository. Successively, involving a group of students, we have checked if the $\mathcal{S}^3$ can effectively support designers in delivering models respecting safeness, soundness and message-relaxed soundness properties, and if $\mathcal{S}^3$ is perceived as a usable and useful tool by its final users. Finally, we have measured the $\mathcal{S}^3$ performances to assess its suitability in relation to its possible adoption in practice. In particular, we have compared the performances of $\mathcal{S}^3$ with those of a widely used tool, that is LoLA (Schmidt, 2000).

The rest of the paper is organised as follows. Section 2 discusses the motivations underlying our work. Section 3 provides the formal framework at the basis of our approach. Section 4 shows how the formal concepts have been realised in our verification tool $\mathcal{S}^3$, while Section 5 presents the results of the experiments we have carried out to validate both the formal framework and the tool. Finally, Section 6 discusses related works and Section 7 concludes the paper.

2. Motivations

This section provides some basic notions on the BPMN standard (OMG, 2011) and on the main elements that constitute the notation. Successively we discuss some simple scenarios to better highlight the possible issues related to the combined usage of sub-process elements and message exchange within a model. Finally, we introduce a case study that we then use for illustrative purposes.

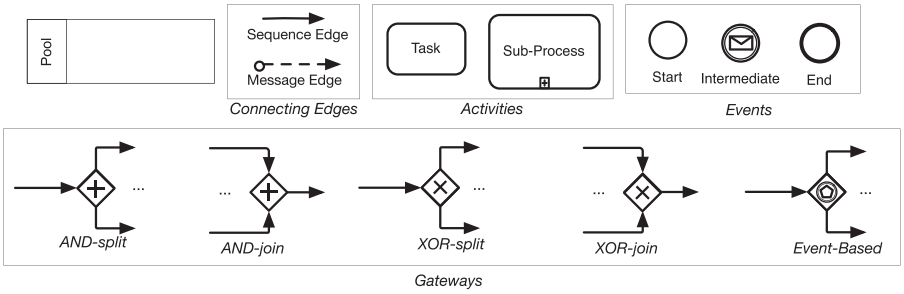


Fig. 1. BPMN Collaboration Elements.

2.1. BPMN Overview

In the last years BPMN has acquired a clear relevance among the notations used to model business processes both in academia and industry, and then to implement supporting information systems. The notation is extremely rich but, as shown in Muehlen and Recker (2008), only less than 20% of its elements is regularly used in designing business process models. Our study has been currently restricted to such a subset as illustrated in Fig. 1. The meaning of the considered elements is explained in the following.

Pools are used to represent the participants (organisations, offices, etc.) involved in a collaboration, and they include details on internal process specifications. **Connecting Edges** are used to connect BPMN elements. In particular, a *Message Edge* is a dashed connector used to visualise communication flows between organisations, while a *Sequence Edge* is a solid connector used to specify the internal execution flow of a process. **Activities** are used to represent specific activities to be performed within a process. A *task* is an atomic activity for which further details are not provided. It can also be used to send and receive messages to another pool. A *sub-process* is used to represent an activity that can be broken down to a finer level of detail. The use of such element can improve understandability, as it permits to relate different level of abstractions in a process model. **Events**, drawn as circles, are used to represent something that can happen. A *Start Event* represents the point from which the process starts, an *Intermediate Event* represents something that happens during process execution, an *End Event* represents the process termination. To improve the understandability of models, we add labels to end events when they are more than one in the same process. When an intermediate event is source or target of a message edge, it is called *Message Event*. According to the different kinds of message edge connections, we can observe the following situations: (i) *Throw Intermediate Event* is an intermediate event with an outgoing message edge; the event element sends a message; (ii) *Catch Intermediate Event* is an intermediate event with an incoming message edge; the event element receives a message. **Gateways** are used to manage the control flow of a process. Gateways act as either join nodes (merging incoming sequence edges) or split nodes (forking into outgoing sequence edges). As best practice, in our work we enforce that a gateway cannot have both multiple input and multiple output flows; such behaviour can be achieved by two gateways in sequence to first converge and then diverge the flows. Different types of gateways are available. An *AND gateway* permits to manage parallel execution flows. An AND-split gateway is used to model the parallel activation of two or more branches, as all outgoing sequence edges are started simultaneously. An AND-join gateway synchronises the execution of two or more parallel branches, as it waits for all incoming sequence edges to complete before triggering the outgoing flow. A *XOR gateway* gives the possibility to describe internal choices. In particular, a XOR-split gateway is used to introduce branches to be selected on the base of exclusive conditions. When executed, the gateway activates only one outgoing edge corresponding to the satisfied condition. A XOR-join gateway acts as a pass-through, meaning that its outgoing edge is activated each time the gateway is reached. An *Event-Based gateway* is similar to the XOR-split gateway, but the activation of its outgoing branches depends on the occurrence of a set of catching events. Basically, such events are in a race condition: the first event that is triggered wins the race and disables all the other ones.

The execution of BPMN models is based on the notion of *token*. This is a theoretical concept used as an aid to define the behaviour of processes, and hence collaborations, that are being performed (OMG, 2011, Section 7.1.1). Roughly, a token is generated by a start event, traverses the sequence edges of a process and passes through its elements enabling their execution, and finally it is consumed by an end event. In a collaboration, the process execution can trigger the exchange of messages.

2.2. On correctness of BPMN collaborations

The verification of BPMN diagrams is a widely investigated domain (Morimoto, 2008; Groefsema and Bucur, 2013; Fellman and Zasada, 2016). However, to the best of our knowledge, there are no comprehensive studies on how correctness is impacted by the introduction of message exchanges, sub-process elements, and their combined usage during the verification phase.

As specified in the standard (OMG, 2011, Section 8.3.2, pp.74), **message exchanges** in collaboration diagrams are assumed to behave accordingly to an asynchronous paradigm. This allows the sending partner to avoid waiting for the reception of a sent message. Asynchronous communication may lead to collaborative scenarios that do not satisfy the soundness property (requiring all processes involved in a collaboration to successful complete), because one or more messages remain pending while the whole collaboration somehow completes. However, the occurrence of such a situation should not be necessarily considered as an error, but its assessment should be better left to the model designer.

To clarify the point, three simple examples from the manufacturing domain are proposed in Fig. 2. In Fig. 2 (A) we consider a scenario where the CEO of a manufacturing company needs that either the marketing manager or the production manager joins him at a meeting with potential customers. While the marketing manager always answers to the request by providing his availability, the production manager may not answer in case he is too busy with the management of the production line. It is irrelevant for the CEO who will join him at the meeting: he will simply select for this duty the manger that replies first. This is expressed in the model by means of an event-based gateway. Indeed, the reception of a message from a manager has two effects: (i) it disables the reception of a message from the other manager; (ii) it triggers a series of activities for sending the meeting information to the manager who answered first, and cancelling the assistance request to the other one. It is worth noticing that, due to the race-condition

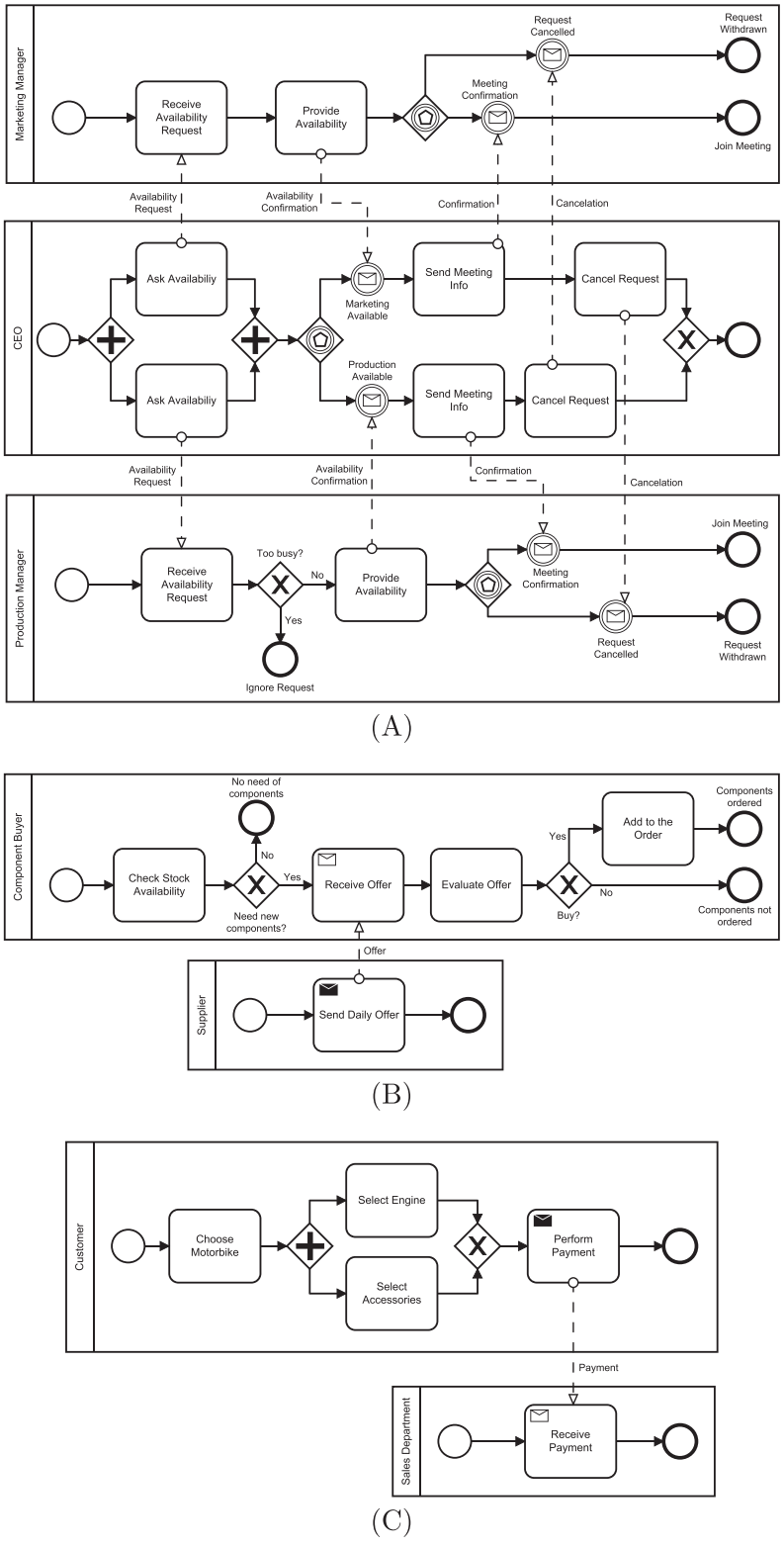


Fig. 2. BPMNModels with Pending Messages.

nature of the event-based gateway semantics, once a message from a manager is received by the CEO, the possibly subsequent message from the other manager is ignored and remains pending. Anyway, this is completely acceptable in this scenario. A similar situation can be observed in Fig. 2 (B), depicting the collaboration between a component buyer and a supplier for the order of components whose price changes on a daily basis. Every working day, the buyer checks the stock availability for a given kind of components and, in case of need, evaluates the offer that each day the related supplier provides him. If the offer is positively evaluated, the order for the needed amount of components is added to the next order to that supplier (the subsequent order management activities are

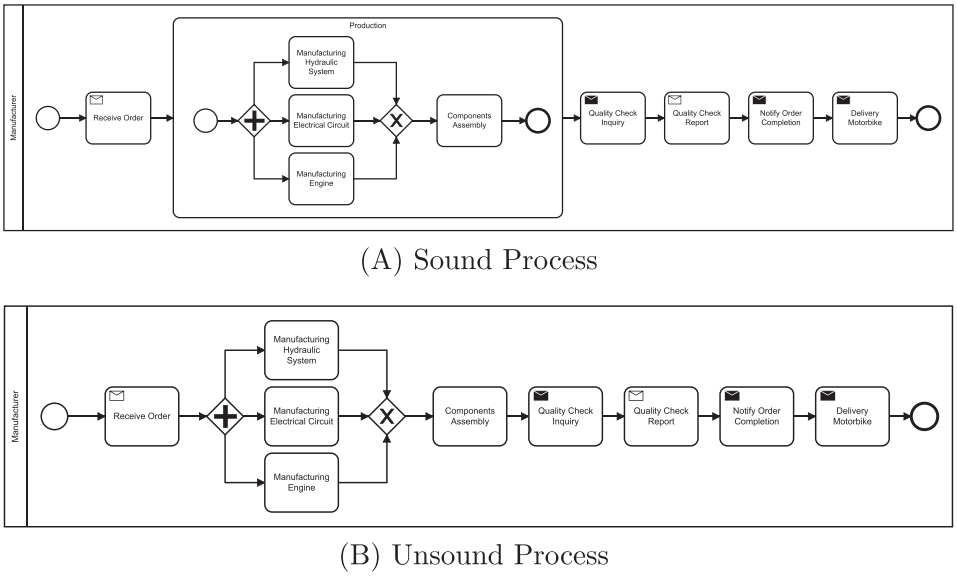


Fig. 3. BPMN Manufacturer Models.

omitted here for the sake of simplicity). In case the buyer does not need to purchase new components, a pending message from the supplier will be left in the buyer's queue. Again, in this situation the presence of a pending message does not affect the correct completion of the collaboration, as the offer can be simply ignored by the buyer when he is not interested in new components.

Finally, Fig. 2 (C) shows a scenario where, differently from the previous two examples, the presence of pending messages at the collaboration completion is not desirable. We consider a customer of a motorbike manufacturer that, after the choice of the motorbike model, selects the engine and the accessories, and then concludes the purchase with the payment directed to the sales department of the motorbike manufacturing company. Notably, the choices of engine and accessories can be done in any order. However, in this BPMN model, the *Perform Payment* task is executed every time one of such choices is done, resulting in a double sending of the payment message. Even if only one out of the two messages will be effectively consumed by the sales department, which prevents a double payment, this model behaviour does not correctly represent the real situation and, hence, it has to be considered as a modelling error. It can be simply fixed by replacing the XOR join gateway in the customer process by an AND join gateway; in this way, the payment is performed only once, because the corresponding task is activated only when both the engine and the accessories are selected.

The property of soundness is the most commonly requested quality criteria for business processes, since it considers both control-flow and message-flow for ensuring correctness. However, in some cases it may result too strict for evaluating a collaboration, in particular when message exchanges and sub-processes are used. To better handle these scenarios, we propose here the "message-relaxed soundness" property, which is a relaxed version of soundness that permits to distinguish those situations where the violation of the soundness property is only due to possible issues related to message exchanges. These issues have to be considered as warnings by the designer, who can decide to leave the model *as-it-is*, as they are due to messages that can be ignored (see Fig. 2 (A)-(B)), or to fix the inconsistencies (see Fig. 2 (C)).

In relation to the **sub-process element**, the BPMN standard defines it as *an activity that encapsulates a process that is in turn modelled by activities, gateways, events, and sequence flows*(OMG, 2011, Section 13.2.4, pp.430). Once activated, a sub-process instance remains in the execution state as long as the encapsulated process is running. Then, only when none of its internal activity is active the sub-process completes (OMG, 2011, pp. 431), and as a result the control is passed to the sequence edge outgoing from the sub-process element. Therefore, according to the standard, a collapsed sub-processes can be considered as a "normal" task consuming an incoming token when it starts, and then it produces just an outgoing token when it ends. For this reason, the sub-process element cannot be considered just syntactic sugar, hence it is not possible to perform verification activities just flattening the model, i.e. by simply substituting the sub-process element by its internal content.

To illustrate such a situation we resort again to models from the manufacturing domain. We consider now a company that produces and assemblies motorbike components for different brands. Whenever the company receives an order from the customer, it analyses the request and activates the production process. Such a process is composed of three parallel activities referring to the production of three main components of the motorbike: the hydraulic system, the electrical circuit, and the engine. As soon as any of these components is produced, it is immediately assembled with the rest of the motorbike. Notice that, for security reason, only one type of component can be assembled at the same time. When the assembly is fully completed, and hence the production process is terminated, the motorbike is ready for the quality check. This check is performed by an autonomous participant acting as an external third party. The quality check procedure consists in performing a quality test and in the preparation of the resulting report, sent to the company that verifies the result. Then, the company notifies the order completion to the customer and delivers to him the motorbike. Fig. 3 shows two BPMN models of such a scenario; for the sake of presentation, only the manufacturer participant is reported, the other ones will be introduced in the collaboration later. In the process of the manufacturer participant specified in Fig. 3 (A) the production phase is embedded into a sub-process, while Fig. 3 (B) shows the same process where the sub-process has been flattened into the main process. Notably, differently from what could be expected, according to the definition provided by the standard the two processes lead to different results when the soundness property is checked. In particular, even if the sub-process in Fig. 3 (A) is unsound, the overall process is sound, since only one token is propagated after the execution of the sub-process, which then en-

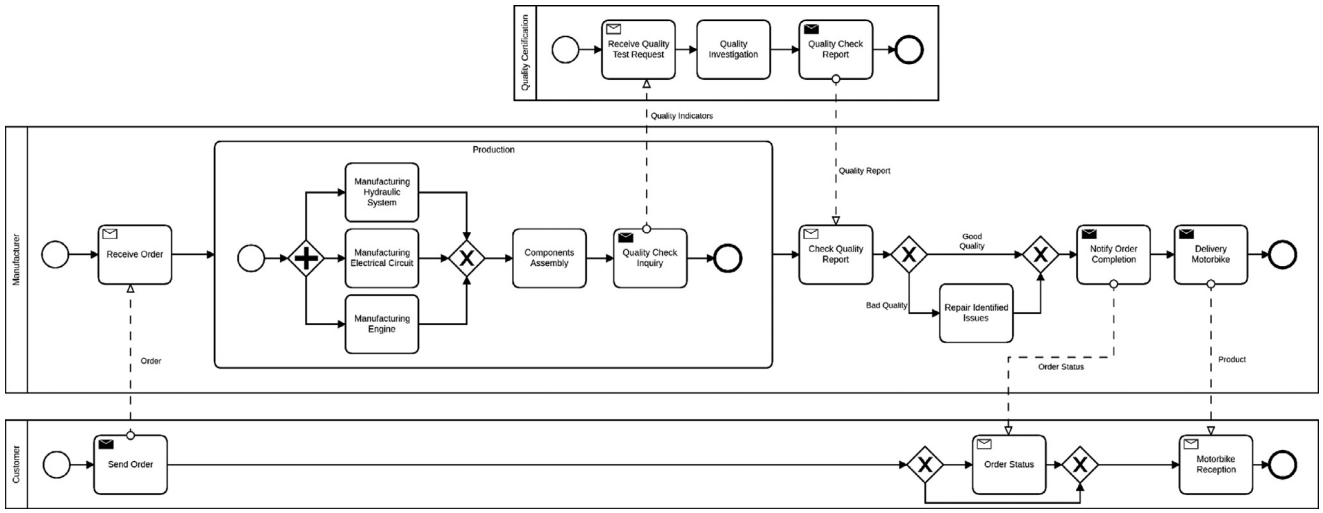


Fig. 4. Order Management (Message-Relaxed Sound Collaboration).

sures the proper completion of the process. A different result can be observed for the flattened process model in Fig. 3 (B). In this case we are in front of an unsound process, since the two tokens generated by the AND-split gateway will both reach the end event, affecting then the proper completion of the overall process. It is worth noticing that the model fragment starting with an AND-split gateway and ending with a XOR-join gateway, which duplicates the incoming token, is a natural way to model the parallel execution of the manufacturing activities that must be followed by the assembly task, which according to the requirements cannot be parallelised but separately executed for each produced component.

Finally, to better illustrate the subtle effects possibly resulting from the **interplay of sub-processes and message exchanges**, we extend the manufacturing model by including the other participants involved in the order management collaboration. The model in Fig. 4 shows the interactions of the manufacturer with a customer and a quality certification office.

Focussing on the customer, he sends the order to the manufacturer and then, before receiving the bike, he can possibly receive and check the order completion notification. The process of the manufacturer is similar to the one previously described, but this time the sub-process includes the request for quality checking within the sub-process. This is a modelling error leading to the sending of a new request for quality verification for each assembly step, which is undesired according to the requirements of this scenario. The effect on the resulting model is that the unsoundness of the process included in the sub-process element is brought outside, causing the request of many activations of the quality certification process. In addition, also the notify order completion message will not be always received by the customer. This could be due, for instance, to the fact that the customer could deliver the order using different channels (e.g., web portal or mobile app), one of which does not permit to check the status of the order. The considered situation is quite common when one of the partners in a collaboration evolves acquiring the capability of sending additional information. However, to still ensure interoperability among the interacting partners, the loss of such additional messages is typically not judged as erroneous.

Summing up, the scenario in Fig. 4 includes two different situations where the soundness property is violated due to issues related to message exchanges. However, they are judged differently: negatively for the first case, since it corresponds to an unintended behaviour, while for the second case the loss of a message can be acceptable. Fig. 5 reports a revised version of the collaboration in which the highlighted issue is solved by bringing the sending task outside of the sub-process. Notably, the collaboration is still not sound, but it satisfies the message relaxed soundness and, in fact, it can be considered adequate by the designer.

The presented examples are intentionally kept simple, as they have been conceived to clarify the motivations of our work. Anyway, in real contexts these situations can easily arise, especially when complex scenarios are considered. To support this statement, we show in Fig. 6 a possible extension of the manufacturing collaboration with further details, which make the model richer (as it consists of more than 50 elements) and more realistic. Roughly, besides detailing the involved processes with further tasks, we introduce another participant to the collaboration devoted to deal with the delivery of motorbikes. On the one hand, this model provides evidence of the applicability of our approach to concrete scenarios. On the other hand, it shows that identifying possibly undesired behaviours, even for experienced designers, may become increasingly difficult when considering larger and more complex models. Indeed, despite at a first glance the model may seem (message-relaxed) sound, it actually is unsound. The unsoundness is due to a deadlock in the customer process: while the manufacturer only sends a single notification message, now the customer can check the order status more than once (see the loop around the *Check Order Status* task), and hence he may remain blocked waiting for an order status message. The error handling can be fixed by removing this looping behaviour in the customer, or, if the intention of the designer is to keep it, the manufacturer process can be extended in order to send more order status messages before the order completion one.

3. The formal framework

This section presents the formal framework at the basis of our correctness checking approach for BPMN models and, hence, of the related $\mathcal{S}^3$ tool. We first present the syntax and operational semantics we defined for a subset of BPMN elements. We then exploit this formal characterisation to define the notions of safeness and (two variants of) soundness for BPMN collaborations.

The considered subset of BPMN elements includes those elements needed to describe message exchanges and sub-processes, which play a key role in our work while being usually not considered or over-abstracted by other formalisations (e.g. El-Saber and Boronat, 2014; Dijkman et al., 2008). In selecting the other elements, following a pragmatic approach, we focused on those reg-

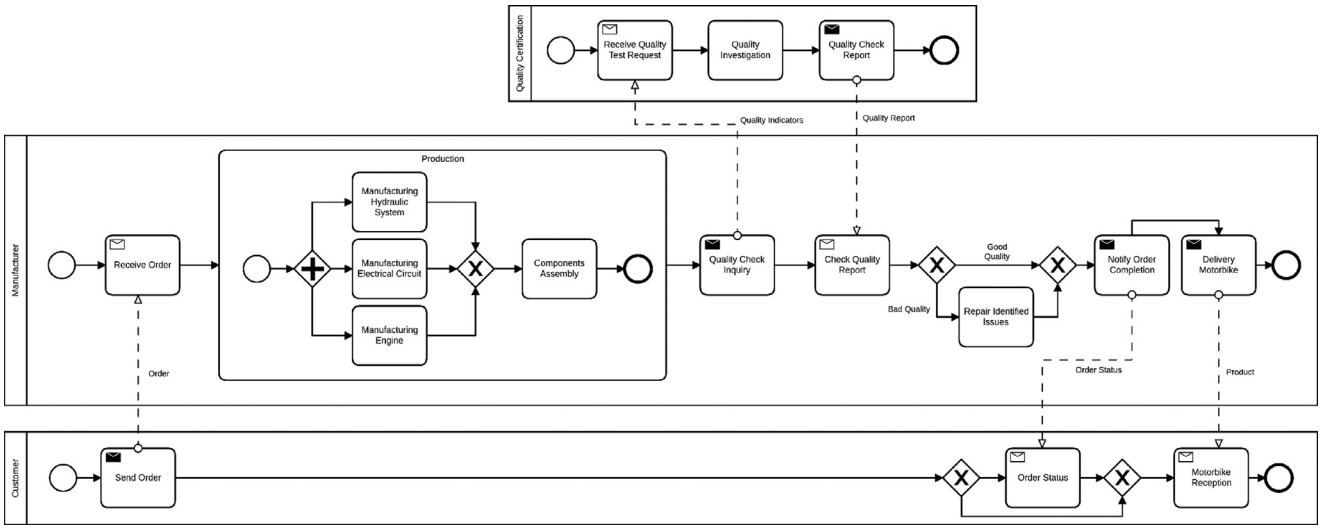


Fig. 5. Order Management - Revised (Message-Relaxed Sound Collaboration).

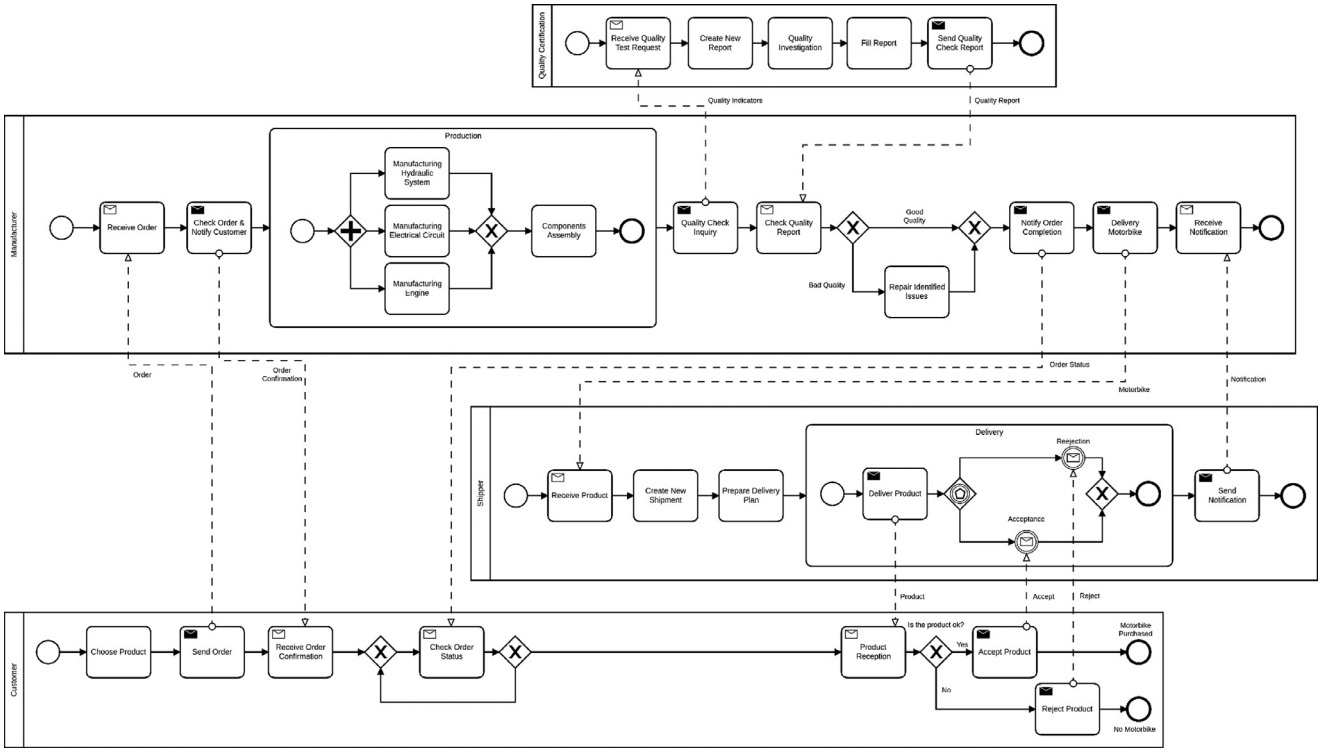


Fig. 6. Order Management - Extension.

ularly used in practice to design process models (Muehlen and Recker, 2008). We have hence left out those features of BPMN whose formal treatment is orthogonal to the addressed problem. In particular, we have not considered such aspects and constructs as inclusive gateways, timed events, error handling, data objects, and multiple instances.

To better illustrate the technicalities of our formal framework we use as a running example the model from the manufacturing domain shown in Fig. 2 (B).

3.1. Syntax of BPMN collaborations

To enable the formal treatment of the collaborations' semantics we resort to a textual representation of BPMN elements. In particular, we defined a BNF syntax of their model structure in Fig. 7. In the proposed grammar, the non-terminal symbol C represents a *Collaboration Structure*, while the terminal symbols, denoted by the sans serif font, are the considered BPMN elements, i.e. events, tasks, sub-processes and gateways. Even if the reader could notice that our syntax is too permissive with respect to the BPMN notation, we will consider only collaborations that are admitted by the standard. Indeed, we are not proposing an alternative modelling notation, but we are only using a textual representation of BPMN models, which is more manageable for writing operational rules than the graphical notation. Therefore, in our analysis we will only consider terms of the syntax that are derived from BPMN models produced by using a BPMN graphical modelling environment such as Camunda.

$$
\begin{array}{rcl}
C & ::= & \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \mid \mathsf{startRcv}(\mathsf{m}, \mathsf{e}_o) \mid \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \mid \mathsf{andSplit}(\mathsf{e}_i, E_o) \\
 & \mid & \mathsf{xorSplit}(\mathsf{e}_i, E_o) \mid \mathsf{andJoin}(E_i, \mathsf{e}_o) \mid \mathsf{xorJoin}(E_i, \mathsf{e}_o) \\
 & \mid & \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh})) \mid \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o) \mid \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \\
 & \mid & \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \\
 & \mid & \mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o) \mid C \| C
\end{array}
$$

Fig. 7. Syntax of BPMN Collaboration Structures.

In the following, $\mathbb{E}$ denotes the set of *sequence edges*, $\mathsf{e} \in \mathbb{E}$ a sequence edge, while $E \in 2^{\mathbb{E}}$ a set of edges. For reader's convenience, we refer with e_i to the edge incoming into an element and with e_o the edge outgoing from an element. In the edge set $\mathbb{E}$ we also include spurious edges for denoting the enabled status of start events and the complete status of end events, named *enabling* and *completing* edges, respectively. They are needed to guarantee activation of sub-processes as well as to check their completion. Moreover, $\mathbb{M}$ denotes the set of *message edges*, and $\mathsf{m} \in \mathbb{M}$ denotes a message edge, allowing message exchanges between pairs of participants in the collaboration. The correspondence between the syntax used here and the graphical notation of BPMN is straightforward, we just highlight below the key points for the study carried out in this paper.

- $\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)$ represents a start event that can be activated by means of the enabling edge e_{enb} and has an outgoing edge e_o.
- $\mathsf{startRcv}(\mathsf{m}, \mathsf{e}_o)$ represents a message start event that can be activated by means of the receiving of a message m and has an outgoing edge e_o.
- $\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$ represents an end event with an incoming edge e_i and a completing edge e_{cmp}.
- $\mathsf{andSplit}(\mathsf{e}_i, E_o)$ (resp. $\mathsf{xorSplit}(\mathsf{e}_i, E_o)$) represents an AND (resp. XOR) split gateway with incoming edge e_i and outgoing edges E_o.
- $\mathsf{andJoin}(E_i, \mathsf{e}_o)$ (resp. $\mathsf{xorJoin}(E_i, \mathsf{e}_o)$) represents an AND (resp. XOR) join gateway with incoming edges E_i and outgoing edge e_o.
- $\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh}))$ represents an event-based gateway with incoming edge e_i and a list of message edges with the related outgoing edges e_{oj} that are enabled by message reception.
- $\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)$ represents a non-communicating task with incoming edge e_i and outgoing edge e_o; communicating task $\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ (resp. $\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$) also receives (resp. sends) a message m.
- $\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ (resp. $\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$) represents an intermediate receiving (resp. sending) event with an incoming edge e_i and an outgoing edge e_o that are able to receive (resp. send) a message m.
- $\mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o)$ represents a sub-process element with incoming edge e_i and outgoing edge e_o. When activated, the enclosed process C behaves according to the elements it consists of, including nested sub-process elements.
- $C||C$ represents a composition of elements in order to render a collaboration structure in terms of a collection of elements.

To achieve a compositional definition, each sequence (resp. message) edge of the BPMN model is split in two parts: the part outgoing from the source element and the part incoming into the target element. The two parts are correlated since edge names in the BPMN model are unique.

Example 1. Let us consider the BPMN model in Fig. 2 (B). The textual representation of its structure is as follows (for reader's convenience, we use e^b_i and e^s_j, with i and j natural numbers, to denote sequence edges of the component buyer and the supplier, respectively):

$$
\begin{array}{l}
\mathsf{start}(\mathsf{e}^b_1, \mathsf{e}^b_2) \,\|\, \mathsf{task}(\mathsf{e}^b_2, \mathsf{e}^b_3) \,\|\, \mathsf{xorSplit}(\mathsf{e}^b_3, \{\mathsf{e}^b_4, \mathsf{e}^b_5\}) \,\|\, \mathsf{end}(\mathsf{e}^b_4, \mathsf{e}^b_6) \\
\quad \|\, \mathsf{taskRcv}(\mathsf{e}^b_5, \mathsf{Offer}, \mathsf{e}^b_7) \,\|\, \mathsf{task}(\mathsf{e}^b_7, \mathsf{e}^b_8) \,\|\, \mathsf{xorSplit}(\mathsf{e}^b_8, \{\mathsf{e}^b_9, \mathsf{e}^b_{10}\}) \\
\quad \|\, \mathsf{end}(\mathsf{e}^b_9, \mathsf{e}^b_{11}) \,\|\, \mathsf{task}(\mathsf{e}^b_{10}, \mathsf{e}^b_{12}) \,\|\, \mathsf{end}(\mathsf{e}^b_{12}, \mathsf{e}^b_{13}) \\
\| \\
\mathsf{start}(\mathsf{e}^s_1, \mathsf{e}^s_2) \,\|\, \mathsf{taskSnd}(\mathsf{e}^s_2, \mathsf{Offer}, \mathsf{e}^s_3) \,\|\, \mathsf{end}(\mathsf{e}^s_3, \mathsf{e}^s_4)
\end{array}
$$

To conveniently refer to the start events of a collaboration C we resort to function $start(C)$, which returns the set of the enabling edges of C:

$$start(C_1 \,\|\, C_2) = start(C_1) \cup start(C_2)$$

$$start(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)) = \{\mathsf{e}_{enb}\} \qquad start(C) = \emptyset \;\text{ for any } C \neq \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)$$

The above function applied on the whole collaboration will return as many edges as the number of start events involved in the collaboration (as the enabling edges of nested sub-processes are ignored), while the application of the function to a process/sub-process returns only one edge. We similarly define the function $end(C)$ on the structure of collaborations in order to refer to end events of C:

$$end(C_1 \,\|\, C_2) = end(C_1) \cup end(C_2)$$

$$end(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})) = \{\mathsf{e}_{cmp}\} \qquad end(C) = \emptyset \;\text{ for any } C \neq \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$$

Example 2. Let C_{bs} be the collaboration structure defined in Example 1. Its start and end events can be referred by the edges identified as follows:

$$start(C_{bs}) = \{\mathsf{e}^b_1, \mathsf{e}^s_1\} \qquad end(C_{bs}) = \{\mathsf{e}^b_6, \mathsf{e}^b_{11}, \mathsf{e}^b_{13}, \mathsf{e}^s_4\}$$

3.2. Semantics of BPMN collaborations

The syntax presented so far permits to describe the mere structure of a collaboration. To describe its semantics we need to enrich it with a notion of execution state, defining the current marking of sequence and message edges. We call *collaboration configuration* this stateful description.

Formally, a configuration has the form $\langle C, \sigma, \delta \rangle$, where: C is a collaboration structure; σ is the first part of the execution state, storing for each sequence edge the current number of tokens marking it; and δ is the second part of the execution state, storing for each message edge the current number of message tokens marking it. Specifically, a state $\sigma : \mathbb{E} \to \mathbb{N}$ is a function mapping edges to numbers of tokens ($\mathbb{N}$ is the set of natural numbers). The state obtained by updating in the state σ the number of tokens of the edge e to n, written as $\sigma \cdot \{\mathsf{e} \mapsto \mathsf{n}\}$, is defined as follows: $(\sigma \cdot \{\mathsf{e} \mapsto \mathsf{n}\})(\mathsf{e}')$ returns n if $\mathsf{e}' = \mathsf{e}$, otherwise it returns $\sigma(\mathsf{e}')$. Moreover, $\delta : \mathbb{M} \to \mathbb{N}$ is a function mapping message edges to numbers of message tokens; thus, $\delta(\mathsf{m}) = \mathsf{n}$ means that there are n messages of type m sent by a participant to another

one that have not been consumed yet. Update of δ is defined in a way similar to the σ's definition. In the *initial state* of a collaboration, the start event of each process in the collaboration must be enabled, i.e. it has a token in its enabling edge, while all other sequence edges (included the enabling edges for the activation of sub-processes) and message edges must be unmarked.

Definition 1 (Initial state of collaboration)**.** Let C be a collaboration, the collaboration configuration $\langle C, \sigma, \delta\rangle$ is the initial one, i.e. predicate $isInit(\langle C, \sigma, \delta\rangle)$ holds, if $\forall\ \mathsf{e}_{enb} \in start(C).\sigma(\mathsf{e}_{enb}) = 1$, $\forall\ \mathsf{e} \in \mathbb{E}\backslash start(C).\sigma(\mathsf{e}) = 0$, and $\forall\ \mathsf{m} \in \mathbb{M}.\delta(\mathsf{m}) = 0$.

Example 3. Let C_{bs} be the collaboration structure defined in Example 1. The initial configuration of the collaboration is $\langle C_{bs}, \sigma_0, \delta_0\rangle$, where:

$$\sigma_0(\mathsf{e}^b_1) = \sigma_0(\mathsf{e}^s_1) = 1$$
$$\sigma_0(\mathsf{e}^b_i) = \sigma_0(\mathsf{e}^s_j) = 0 \quad \forall i, j \neq 1$$
$$\delta_0(\mathsf{Offer}) = 0$$

The operational semantics is defined by means of a *labelled transition system* (LTS). In our case, this is a triple $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ where: $\mathcal{C}$, ranged over by $\langle C, \sigma, \delta\rangle$, is a set of collaboration configurations; $\mathcal{L}$, ranged over by l, is a set of *labels* (of transitions that collaboration configurations can perform); and $\rightarrow \subseteq \mathcal{C} \times \mathcal{L} \times \mathcal{C}$ is a *transition relation*. Labels l represent computational steps: $!\mathsf{m}$ and $?\mathsf{m}$ denote sending and receiving actions, respectively, while E denotes the set of edges from which a token is moved, thus permitting to identify the current position in the execution flow (for the sake of readability, we write the set $\{\mathsf{e}\}$ as e). We will write $\langle C, \sigma, \delta\rangle \xrightarrow{l} \langle \sigma', \delta'\rangle$ to indicate that $(\langle C, \sigma, \delta\rangle, l, \langle C, \sigma', \delta'\rangle) \in \rightarrow$ and say that 'the collaboration in the configuration $\langle C, \sigma, \delta\rangle$ can do a transition labelled l and become the collaboration configuration $\langle C, \sigma', \delta'\rangle$ in doing so'. Notice that we omit the collaboration structure from the target configuration of the transition, since collaboration execution only affects the current states and not the collaboration structure. Notably, despite the presence of labels, this has to be thought of as a reduction semantics, because labels are not used for synchronisation (as instead it usually happens in labeled semantics), but only for keeping track of the performed actions in order to enable the verification.

Before introducing the semantic rules, we define the auxiliary functions they exploit. Specifically, function $inc : \mathbb{S} \times \mathbb{E} \rightarrow \mathbb{S}$ (resp. $dec : \mathbb{S} \times \mathbb{E} \rightarrow \mathbb{S}$), where $\mathbb{S}$ is the set of states, allows updating a state by incrementing (resp. decrementing) by one the number of tokens marking an edge in the state. Formally, they are defined as follows: $inc(\sigma, \mathsf{e}) = \sigma \cdot \{\mathsf{e} \mapsto \sigma(\mathsf{e}) + 1\}$ and $dec(\sigma, \mathsf{e}) = \sigma \cdot \{\mathsf{e} \mapsto \sigma(\mathsf{e}) - 1\}$. These functions extend in a natural way to sets of edges as follows: $inc(\sigma, \emptyset) = \sigma$ and $inc(\sigma, \{\mathsf{e}\} \cup E)) = inc(inc(\sigma, \mathsf{e}), E)$; the cases for *dec* are similar. The functions *inc* and *dec* for δ are defined in a way similar to σ's definitions. We also use the function $zero : \mathbb{S} \times \mathbb{E} \rightarrow \mathbb{S}$ that allows updating a state by setting to zero the number of tokens marking an edge in the state. Formally, it is defined as follows: $zero(\sigma, \mathsf{e}) = \sigma \cdot \{\mathsf{e} \mapsto 0\}$. Also in this case the function extends in a natural way to sets of edges as follows: $zero(\sigma, \emptyset) = \sigma$ and $zero(\sigma, \{\mathsf{e}\} \cup E)) = zero(zero(\sigma, \mathsf{e}), E)$.

Example 4. Let us consider the state σ_0 in Example 3. The state σ_1 obtained by moving the token from the edge e^b_1 to the edge e^b_2 is obtained as follows: $\sigma_1 = inc(\sigma'_0, \mathsf{e}^b_2)$ with $\sigma'_0 = dec(\sigma_0, \mathsf{e}^b_1)$.

We also use the function $marked(\sigma, E)$ to refer to the set of edges in E marked by at least one token, which is defined as follows: $marked(\sigma, \emptyset) = \emptyset$, and $marked(\sigma, \{\mathsf{e}\} \cup E)$ returns $\{\mathsf{e}\} \cup marked(\sigma, E)$ if $\sigma(\mathsf{e}) > 0$, otherwise it returns $marked(\sigma, E)$.

Example 5. Let us consider the state σ_1 in Example 4. The marked edges of the collaboration in this state are as follows:

$$marked(\sigma_1, \{\mathsf{e}^b_1, \ldots, \mathsf{e}^b_{13}, \mathsf{e}^s_1, \ldots, \mathsf{e}^s_4\}) = \{\mathsf{e}^b_2, \mathsf{e}^s_1\}$$

To check the completion of a sub-process, we exploit the boolean predicate $completed(C, \sigma)$ defined as follow.

$$\exists\, \mathsf{e}_{cmp} \in end(C).\mathsf{e}_{cmp} \in marked(\sigma, end(C)) \quad \wedge \quad \forall\, \mathsf{e} \in \mathbb{E}\backslash end(C).\sigma(\mathsf{e}) = 0$$

It requires that at least one token reaches an end node, and there is no more token on the sequence edges of the sub-process; this subsumes that more end nodes can be marked. Notably, the completion of a sub-process does not depend on the exchanged messages, and it is defined considering the arbitrary topology of the model. This definition complies with the prescriptions of the BPMN standard (OMG, 2011, pp. 426, 431).

Example 6. Let us consider again the state σ_1 in Example 4 and the collaboration structure C_{bs} in Example 1. We have that $completed(C_{bs}, \sigma_1) = false$ because all edges e^b_6, e^b_{11}, e^b_{13}, e^s_4 in $end(C_{bs})$ (see Example 2) are unmarked, i.e. they are not in $marked(\sigma_1, \{\mathsf{e}^b_1, \ldots, \mathsf{e}^b_{13}, \mathsf{e}^s_1, \ldots, \mathsf{e}^s_4\})$ (see Example 5).

Now, we can define in Fig. 8 the transition relation over collaboration configurations, which formalises the execution of a collaboration in terms of edge and message marking evolution.

We briefly comment on salient points. Rule *Start* starts the execution of a process/sub-process when it has been activated (i.e., its enabling edge e_{enb} is marked). The effect of the rule is to increment the number of tokens in the edge outgoing from the start event and to decrease the number of tokens in the enabling edge. Rule *End* instead is enabled when there is at least one token in the incoming edge of the end event, which is then moved to the completing edge. Rule *StartRcv* start the execution of a process when there is at least a token in the incoming message. The effect of the rule is to increment the number of tokens in the edge outgoing from the start event and remove the token from the message edge. Rule *AndSplit* is applied when there is at least one token in the incoming edge of an AND split gateway; as result of its application the rule decrements the number of tokens in the incoming edge and increments those in each outgoing edge. Similarly, rule *XorSplit* is applied when a token is available in the incoming edge of a XOR split gateway, the rule moves the token from the incoming edge to one of the outgoing edges, non-deterministically chosen. Rule *AndJoin* decrements the tokens in each incoming edge and increments the number of tokens of the outgoing edge, when each incoming edge has at least one token. Rule *XorJoin* is activated every time there is a token in one of the incoming edges, which is then moved to the outgoing edge. Rule *EventBased* is activated when there is a token in the incoming edge and there is a message m_j to be consumed, so that the application of the rule moves the token from the incoming edge to the outgoing edge corresponding to the received message, whose number of message tokens is decreased (i.e., a message from the corresponding queue is consumed). Rule *Task* deals with simple tasks, acting as a pass through. It is activated only when there is a token in the incoming edge, which is then moved to the outgoing edge. Rule *TaskRcv* is activated when there is a token in the incoming edge and, in addition, there is a message to be consumed. Similarly, rule *TaskSnd* sends a message before moving the token to the outgoing edge. Rule *InterRcv* (resp. *InterSnd*) follows the same behavior of rule *TaskRcv* (resp. *TaskSnd*). Rules $SubP_{start}$, $SubP_{run}$ and Sub_{end} deal with the sub-process element. Rule $SubP_{start}$ is activated only when there is a token in the incoming edge of the sub-process, which is then moved to the enabling edge of the start event in the sub-process body. Then, by applying $SubP_{run}$, the sub-process behaves as its body till it completes, according to the completion check performed by the rule $SubP_{end}$. When this last rule is applied, it removes all tokens from

$$\langle \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{\mathsf{e}_{enb}} \langle inc(dec(\sigma, \mathsf{e}_{enb}), \mathsf{e}_o), \delta\rangle \quad \sigma(\mathsf{e}_{enb}) > 0 \qquad (Start)$$

$$\langle \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma, \delta\rangle \xrightarrow{\mathsf{e}_i} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_{cmp}), \delta\rangle \quad \sigma(\mathsf{e}_i) > 0 \qquad (End)$$

$$\langle \mathsf{startRcv}(\mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{?\mathsf{m}} \langle inc(\sigma, \mathsf{e}_o), dec(\delta, \mathsf{m})\rangle \quad \delta(\mathsf{m}) > 0 \qquad (StartRcv)$$

$$\langle \mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma, \delta\rangle \xrightarrow{\mathsf{e}_i} \langle inc(dec(\sigma, \mathsf{e}_i), E_o), \delta\rangle \quad \sigma(\mathsf{e}_i) > 0 \qquad (AndSplit)$$

$$\langle \mathsf{xorSplit}(\mathsf{e}_i, \{\mathsf{e}_o\} \cup E_o), \sigma, \delta\rangle \xrightarrow{\mathsf{e}_i} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), \delta\rangle \quad \sigma(\mathsf{e}_i) > 0 \qquad (XorSplit)$$

$$\langle \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{E_i} \langle inc(dec(\sigma, E_i), \mathsf{e}_o), \delta\rangle \quad \forall \mathsf{e} \in E_i \;.\; \sigma(\mathsf{e}) > 0 \qquad (AndJoin)$$

$$\langle \mathsf{xorJoin}(\{\mathsf{e}_i\} \cup E_i, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{\mathsf{e}_i} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), \delta\rangle \quad \sigma(\mathsf{e}_i) > 0 \qquad (XorJoin)$$

$$\langle \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh})), \sigma, \delta\rangle \xrightarrow{?\mathsf{m}_j} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_{oj}), dec(\delta, \mathsf{m}_j)\rangle \quad \begin{array}{l} 1 \le j \le h, \\ \sigma(\mathsf{e}_i) > 0, \delta(\mathsf{m}_j) > 0 \end{array} \qquad (EventG)$$

$$\langle \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{\mathsf{e}_i} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), \delta\rangle \quad \sigma(\mathsf{e}_i) > 0 \qquad (Task)$$

$$\langle \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{?\mathsf{m}} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), dec(\delta, \mathsf{m})\rangle \quad \sigma(\mathsf{e}_i) > 0, \delta(\mathsf{m}) > 0 \qquad (TaskRcv)$$

$$\langle \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{!\mathsf{m}} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), inc(\delta, \mathsf{m})\rangle \quad \sigma(\mathsf{e}_i) > 0 \qquad (TaskSnd)$$

$$\langle \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{?\mathsf{m}} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), dec(\delta, \mathsf{m})\rangle \quad \sigma(\mathsf{e}_i) > 0, \delta(\mathsf{m}) > 0 \qquad (InterRcv)$$

$$\langle \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{!\mathsf{m}} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), inc(\delta, \mathsf{m})\rangle \quad \sigma(\mathsf{e}_i) > 0 \qquad (InterSnd)$$

$$\langle \mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{\mathsf{e}_i} \langle inc(dec(\sigma, \mathsf{e}_i), start(C)), \delta\rangle \quad \sigma(\mathsf{e}_i) > 0 \qquad (SubP_{start})$$

$$\langle \mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{marked(\sigma, end(C))} \langle inc(zero(\sigma, end(C)), \mathsf{e}_o), \delta\rangle \quad completed(C, \sigma) \qquad (SubP_{end})$$

$$\frac{\langle C, \sigma, \delta\rangle \xrightarrow{l} \langle \sigma', \delta'\rangle}{\langle \mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{l} \langle \sigma', \delta'\rangle} (SubP_{run})$$

$$\frac{\langle C_1, \sigma, \delta\rangle \xrightarrow{l} \langle \sigma', \delta'\rangle}{\langle C_1 \parallel C_2, \sigma, \delta\rangle \xrightarrow{l} \langle \sigma', \delta'\rangle} (Int_1) \qquad \frac{\langle C_2, \sigma, \delta\rangle \xrightarrow{l} \langle \sigma', \delta'\rangle}{\langle C_1 \parallel C_2, \sigma, \delta\rangle \xrightarrow{l} \langle \sigma', \delta'\rangle} (Int_2)$$

Fig. 8. BPMN Collaboration Semantics.

the sub-process, and adds a token to the outgoing edge of the sub-process. Actually, due to the definition of sub-process completion, only the completing edges of the end events within the sub-process body need to be set to zero. Finally, *Int*$_1$ and *Int*$_2$ deal with interleaving in a standard way.

Example 7. Fig. 9 (upper part) shows an excerpt of the LTS representing the semantics of the collaboration C_{bs} (the full LTS consists of 27 states). The execution states σ_i and δ_j of the considered configurations, together with a graphical representation of the corresponding token markings, are reported in the lower part of Fig. 9.

The LTS is obtained by applying the rules in Fig. 8, starting from the initial configuration $\langle C_{bs}, \sigma_0, \delta_0\rangle$. To clarify how the LTS is generated, we describe below how the two transitions from the initial configuration are inferred:

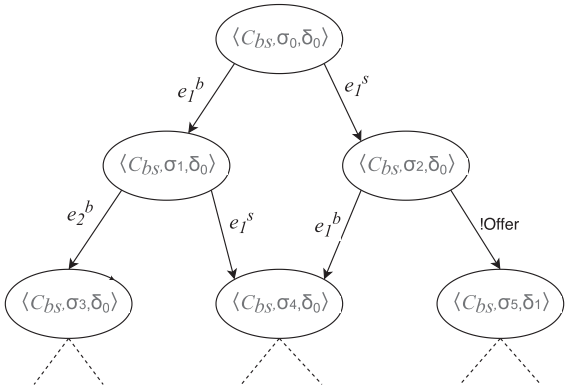
- focussing on the start event of the buyer, the collaboration C_{bs} has the form $\mathsf{start}(\mathsf{e}^b_1, \mathsf{e}^b_2) \parallel C'_{bs}$ (see Example 1), hence by rule (*Int*$_1$) the execution state produced by the transition is the same obtained from $\langle \mathsf{start}(\mathsf{e}^b_1, \mathsf{e}^b_2), \sigma_0, \delta_0\rangle$ by applying rule (*Start*); formally, the transition is inferred as follows:

$$\dfrac{\dfrac{}{\langle \mathsf{start}(\mathsf{e}^b_1, \mathsf{e}^b_2), \sigma_0, \delta_0\rangle \xrightarrow{\mathsf{e}^b_1} \langle \sigma_1, \delta_0\rangle}(Start)}{\langle \mathsf{start}(\mathsf{e}^b_1, \mathsf{e}^b_2) \parallel C'_{bs}, \sigma_0, \delta_0\rangle \xrightarrow{\mathsf{e}^b_1} \langle \sigma_1, \delta_0\rangle}(Int_1)$$

  with $\sigma_1 = inc(dec(\sigma_0, \mathsf{e}^b_1), \mathsf{e}^b_2)$;
- focussing instead on the start event of the supplier, the collaboration C_{bs} has the form $C''_{bs} \parallel \mathsf{start}(\mathsf{e}^s_1, \mathsf{e}^s_2) \parallel C'''_{bs}$ (see Example 1), thus leading to the following inference:

$$\dfrac{\dfrac{\dfrac{}{\langle \mathsf{start}(\mathsf{e}^s_1, \mathsf{e}^s_2), \sigma_0, \delta_0\rangle \xrightarrow{\mathsf{e}^s_1} \langle \sigma_2, \delta_0\rangle}(Start)}{\langle \mathsf{start}(\mathsf{e}^s_1, \mathsf{e}^s_2) \parallel C'''_{bs}, \sigma_0, \delta_0\rangle \xrightarrow{\mathsf{e}^s_1} \langle \sigma_2, \delta_0\rangle}(Int_1)}{\langle C''_{bs} \parallel \mathsf{start}(\mathsf{e}^s_1, \mathsf{e}^s_2) \parallel C'''_{bs}, \sigma_0, \delta_0\rangle \xrightarrow{\mathsf{e}^s_1} \langle \sigma_2, \delta_0\rangle}(Int_2)$$

  with $\sigma_2 = inc(dec(\sigma_0, \mathsf{e}^s_1), \mathsf{e}^s_2)$.



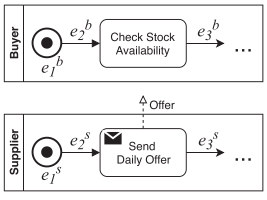
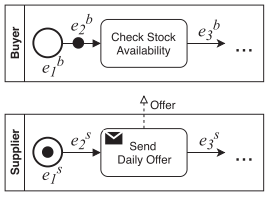
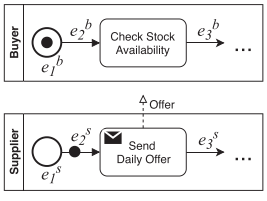
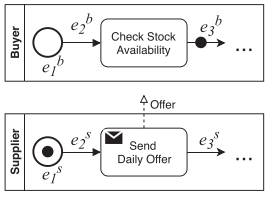
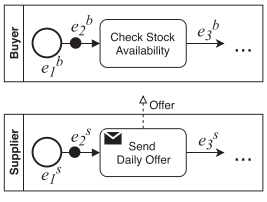
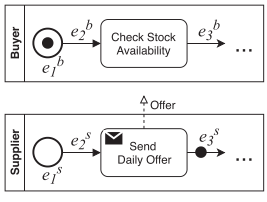
Marking	Execution State	Marking	Execution State
	$\sigma_0(e_1^b) = \sigma_0(e_1^s) = 1$ $\sigma_0(e_i^b) = \sigma_0(e_j^s) = 0$ $\forall i, j \neq 1$ $\delta_0(\text{Offer}) = 0$		$\sigma_1(e_2^b) = \sigma_1(e_1^s) = 1$ $\sigma_1(e_i^b) = \sigma_1(e_j^s) = 0$ $\forall i \neq 2, j \neq 1$ $\delta_0(\text{Offer}) = 0$
	$\sigma_2(e_1^b) = \sigma_2(e_2^s) = 1$ $\sigma_2(e_i^b) = \sigma_2(e_j^s) = 0$ $\forall i \neq 1, j \neq 2$ $\delta_0(\text{Offer}) = 0$		$\sigma_3(e_3^b) = \sigma_3(e_1^s) = 1$ $\sigma_3(e_i^b) = \sigma_3(e_j^s) = 0$ $\forall i \neq 3, j \neq 1$ $\delta_0(\text{Offer}) = 0$
	$\sigma_4(e_2^b) = \sigma_4(e_2^s) = 1$ $\sigma_4(e_i^b) = \sigma_4(e_j^s) = 0$ $\forall i, j \neq 2$ $\delta_0(\text{Offer}) = 0$		$\sigma_5(e_1^b) = \sigma_5(e_3^s) = 1$ $\sigma_5(e_i^b) = \sigma_5(e_j^s) = 0$ $\forall i \neq 1, j \neq 3$ $\delta_1(\text{Offer}) = 1$

Fig. 9. LTS excerpt of the running example (model in Fig. 2 (B)), with associated execution states (and graphical representation of the related marking).

3.3. Safeness and soundness properties

We now provide a formal definition for the correctness properties we verify on BPMN collaboration models. We use below $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$.

Safeness refers to the occurrence of no more than one token at the same time on the same sequence edge of each process along the collaboration execution.

Definition 2 (Safe collaborations)**.** A collaboration C is *safe* if and only if, given σ and δ such that $isInit(\langle C, \sigma, \delta\rangle)$ holds, then for all σ' and δ' such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle \sigma', \delta'\rangle$ we have that $\forall \mathsf{e} \in \mathbb{E}.\sigma'(\mathsf{e}) \leq 1$.

Example 8. The collaboration C_{bs} is safe, because for each collaboration configuration $\langle C_{bs}, \sigma_i, \delta_j\rangle$ reached from $\langle C_{bs}, \sigma_0, \delta_0\rangle$ we have that for any edge e_k^b (resp. e_h^s) in the collaboration it holds that $\sigma_i(\mathsf{e}_k^b) \leq 1$ (resp. $\sigma_i(\mathsf{e}_h^s) \leq 1$).

This trivially holds, because the collaboration structure does not include any AND-split gateway, which is the only construct capable of creating multiple copies of an incoming token.

Soundness is a more elaborated property (we refer to Section 6 for a discussion about different notions of soundness introduced in the literature for various workflow notations, and their relationships with the one proposed here for BPMN collaboration models). Intuitively, the soundness property requires that from any reachable configuration it is possible to reach a configuration where all message queues are empty, and the marked completing edges are marked exactly by a single token while all the other edges are unmarked.

Definition 3 (Soundness)**.** A collaboration C is *sound* if and only if, given σ and δ such that $isInit(\langle C, \sigma, \delta\rangle)$ holds, then for all σ' and δ' such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle \sigma', \delta'\rangle$ we have that there exist σ'' and δ'' such that $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, $\forall \mathsf{e}_{cmp} \in marked(\sigma'', end(C)).\sigma''(\mathsf{e}_{cmp}) = 1$, $\forall \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) = 0$, and $\forall \mathsf{m} \in \mathbb{M}.\delta''(\mathsf{m}) = 0$.

Example 9. The collaboration C_{bs} is unsound, because from the initial configuration it can reach the configuration $\langle C_{bs}, \sigma_i, \delta_j\rangle$ where the buyer participant terminated its execution in the end state labelled by 'No need of components' (i.e., $\sigma_i(\mathsf{e}_6^b) = 1$ and $\sigma_i(\mathsf{e}_k^b) = 0$

for $k \neq 6$), the supplier terminated in its end state (i.e., $\sigma_i(e_4^s) = 1$ and $\sigma_i(e_h^s) = 0$ for $h \neq 4$), but the Offer message sent by the supplier is pending (i.e., $\delta_j(\text{Offer}) \neq 0$).

As mentioned in Section 1 and Section 2, the above definition of soundness may result to be too restrictive (e.g., in presence of event-based gateways or unstructured processes). Therefore, we provide a relaxed variant that does not require message queues to be empty for a proper completion.

Definition 4 (Message-Relaxed Soundness)**.** A collaboration C is *message-relaxed sound* if and only if, given σ and δ such that $isInit(\langle C, \sigma, \delta\rangle)$ holds, then for all σ' and δ' such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle \sigma', \delta'\rangle$ we have that there exist σ'' and δ'' such that $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, $\forall e_{cmp} \in marked(\sigma'', end(C)).\sigma''(e_{cmp}) = 1$, and $\forall e \in \mathbb{E} \backslash end(C).\sigma''(e) = 0$.

Example 10. The collaboration C_{bs} is message-relaxed sound, because as shown in Example 9 the buyer and supplier processes correctly terminate and the presence of a pending message is not relevant for this property.

4. From theory to practice

In this section, we show how we check safeness, soundness and message-relaxed soundness properties on the LTSs generated by the BPMN operational semantics given in Section 3.2. Then, we formally prove the correspondence between these checks and Def. 2, Def. 3 and Def. 4, respectively. Finally, we illustrate the $\mathcal{S}^3$ tool implementing the proposed verification approach.

In the construction of the LTS, each time a new state has to be added we check if a state representing the same collaboration configuration is already present; in such a case, we connect the transition edge under construction to the existing state. In particular, in doing that we consider identical those collaboration configurations that have the same σ and δ ignoring the completing edges. This is motivated by the fact that for the soundness properties it is only relevant that the collaboration completes, regardless which end nodes are marked. This characteristic of the generated LTS is then exploited (see Def. 9) for putting in relation the soundness of the collaboration with the existence in the LTS of a unique "final" state (representing a configuration where all tokens are consumed and no other sequence or message edge is enabled).

We now formally introduce the notion of safeness related to the LTS induced by the semantics.

Definition 5 (Safe LTSs)**.** An LTS $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ of a collaboration is safe if and only if $\forall \langle C, \sigma, \delta\rangle \in \mathcal{C}$ we have that $\forall e \in \mathbb{E} \,.\, \sigma(e) \leq 1$.

The formal definition of soundness requires the definition of the following auxiliary functions determining the incoming labels of a state in the LTS, the presence of an execution trace of the LTS where given labels occur more than once, and the set of edge labels incoming to the end events of a collaboration. We use $\stackrel{l}{\Longrightarrow}$ to denote $\rightarrow^* \stackrel{l}{\rightarrow} \rightarrow^*$.

Definition 6 (Incoming Labels)**.** Let $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ be an LTS and $\langle C, \sigma, \delta\rangle \in \mathcal{C}$, $incoming(C, \sigma, \delta) = \{l \in \mathcal{L} \mid \exists \sigma', \delta' . langle C, \sigma', \delta'\rangle \stackrel{l}{\rightarrow} \langle C, \sigma, \delta\rangle\}$.

Definition 7 (Labels Duplication)**.** Let $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ be an LTS and $L \subseteq \mathcal{L}$ a set of labels, predicate $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle, L)$ holds if $\forall l \in L$ and $\langle C, \sigma_1, \delta_1\rangle, \langle C, \sigma_2, \delta_2\rangle, \langle C, \sigma_3, \delta_3\rangle \in \mathcal{C}$ the sequence $\langle C, \sigma_1, \delta_1\rangle \stackrel{l}{\Longrightarrow} \langle C, \sigma_2, \delta_2\rangle \stackrel{l}{\Longrightarrow} \langle C, \sigma_3, \delta_3\rangle$ never holds.

Definition 8 (End Events Incoming Labels)**.** Let C be a collaboration, then $endIn(\,\cdot\,)$ is inductively defined as follows: $endIn(C_1 \parallel C_2) = endIn(C_1) \cup endIn(C_2)$; $endIn(\text{end}(e_i, e_{cmp})) = \{e_i\}$; and $endIn(C) = \emptyset$ for any $C \neq \text{end}(e_i, e_{cmp})$.

Our notions of soundness on LTSs are defined as follows.

Definition 9 (Sound LTSs)**.** An LTS $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ of a collaboration C is *sound* if and only if $\exists! \langle C, \sigma, \delta\rangle \in \mathcal{C}$ such that:

(i) $\langle C, \sigma, \delta\rangle \not\rightarrow$ (i.e., $\nexists l, \sigma', \delta'$ such that $\langle C, \sigma, \delta\rangle \stackrel{l}{\rightarrow} \langle C, \sigma', \delta'\rangle$);
(ii) $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle, incoming(C, \sigma, \delta))$;
(iii) $incoming(C, \sigma, \delta) = endIn(C)$;
(iv) $\forall e \in \mathbb{E} \backslash end(C).\sigma(e) = 0$;
(v) $\forall m \in \mathbb{M}.\delta(m) = 0$.

Definition 10 (Message-Relaxed Sound LTSs)**.** An LTS $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ of a collaboration C is *message-relaxed sound* if and only if $\forall \langle C, \sigma, \delta\rangle \in \mathcal{C}$ such that $\langle C, \sigma, \delta\rangle \not\rightarrow$ we have that:

(i) $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle, incoming(C, \sigma, \delta))$;
(ii) $incoming(C, \sigma, \delta) = endIn(C)$;
(iii) $\forall e \in \mathbb{E} \backslash end(C).\sigma(e) = 0$.

Now, we show the correspondence of the above definitions with those on collaborations given in Section 3.3. All proofs are reported in the Appendix.

Theorem 1 (Safeness Correspondence)**.** *Let C be a collaboration and $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ its LTS, then C is safe if and only if $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ is safe.*

Theorem 2 (Soundness Correspondence)**.** *Let C be a collaboration and $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ its LTS, then C is sound if and only if $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ is sound.*

Theorem 3 (Message-Relaxed Soundness Correspondence)**.** *Let C be a collaboration and $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ its LTS, then C is message-relaxed sound if and only if $\langle \mathcal{C}, \mathcal{L}, \rightarrow\rangle$ is message-relaxed sound.*

4.1. The $\mathcal{S}^3$ supporting tool

$\mathcal{S}^3$ is an open source software that can be redistributed and eventually modified under the terms of the GPL2 License. The source code as well as the user guide are publicly available, and they can be retrieved from http://pros.unicam.it/s3/.

$\mathcal{S}^3$ is based on a standard client/server architecture. The $\mathcal{S}^3$ server embeds the core of the system and exposes a Java RESTful service. It takes as input a BPMN model in the standard format *.bpmn*, and then returns the results of the properties checking. The received model is parsed using the Camunda APIs, and then the corresponding LTS is derived according to the semantics defined in Section 3.2. The property verification is based on the checks defined in Def. 5, Def. 9 and Def. 10. In case of a property violation, the service returns the corresponding counterexample.

Designers can use the $\mathcal{S}^3$ service via the web application we made available.[2] It is composed by a client, developed in HTML/Javascript, that embeds the Camunda *bpmn.io* modeller. Using the graphical interface depicted in Fig. 10 the model designer can specify its collaboration using all the facilities of the Camunda modeller. Successively, clicking on the button at the top-right corner the verification can be run. The verification results of safeness and soundness will be summarised by a 'traffic light' reporting: the green (resp. red) colour for safeness means that the collaboration model is safe (resp. unsafe); for soundness, instead, the green colour means that the model is sound, the yellow colour corresponds to the warning situation, as the model is unsound but message-relaxed sound (hence, a manual check by the designer is required), finally the red colour means that the model is unsound for both notions. To support the designer in solving issues raised

[2] $\mathcal{S}^3$ web application: http://pros.unicam.it:8080/S3/modeler/.

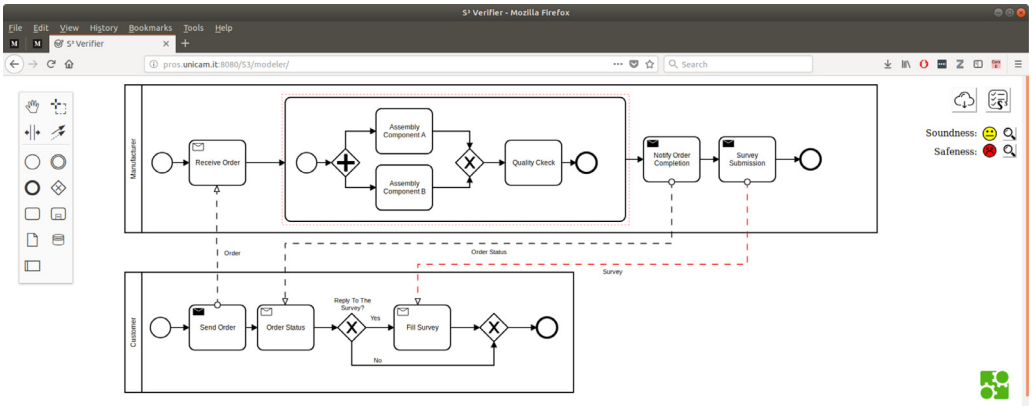


Fig. 10. $\mathcal{S}^3$ Modelling Environment Interface.

by the verification, in case of a negative outcome an additional button depicting a lens gives the possibility to show the corresponding counterexample, which will be displayed directly on the model by colouring the sequence/message edges involved in the violation (see the red-coloured message edge between task "Survey submission" and "Fill survey" in Fig. 10). It is also possible to export the model as a *.bpmn* file.

Finally, $\mathcal{S}^3$ can be also executed locally via a Java stand-alone application we made available.[3] The application allows the user to load a *.bpmn* file to be checked, and hence to verify the considered properties. The graphical interface provides a text area reporting the verification results, and a button to visualise in a separate window the generated LTS.

5. Validation

In this section we report the results of the experiments we carried out in order to validate both the proposed verification framework and the corresponding $\mathcal{S}^3$ tool. In particular, we shaped our validation activity according to the following research questions:

- RQ1 Are safeness, soundness and message-relaxed soundness correctly handled by model designers, or do they release models violating such properties?
- RQ2 Can $\mathcal{S}^3$ effectively support designers in delivering models respecting safeness, soundness and message-relaxed soundness properties?
- RQ3 Is $\mathcal{S}^3$ perceived as usable and useful by its final users?
- RQ4 Are $\mathcal{S}^3$ performances suitable to make possible its adoption in practice?

The following subsections describe the experiments we performed and the conclusions we can derive, as well as possible threats to their validity.

5.1. Evaluation in the large

In this subsection we present the experiments we performed to provide an answer to the research question RQ1. The objective is to have some evidence that the checks we introduce with our approach are needed in practice.

Experiment Set-Up. In order to assess the real value brought by our verification framework we needed to identify a collection of BPMN collaboration models independently defined and not under our control. This avoids that the validation activity could be biased by the objectives of the experiment itself.

To the best of our knowledge, there is no reference dataset of models that is commonly used to validate approaches in the context of BPMN modelling practices. This is, indeed, one of the major challenges nowadays in the BPM community. We have hence identified for such a purpose the BPM Academic Initiative repository (http://bpmai.org/, (Kunze et al., 2012; 2011))[4].

This is a collection of models codified using various process modelling languages. Focussing on BPMN, the raw dataset consists of 1 6032 models, but we restricted to the latest revision of each model having 100% of connectedness[5]. A model without this level of connectedness includes disconnected fragments, which typically could mean that the model has not been finalised, yet. Finally, we restricted our study to collaboration models, for which the introduced properties definitions are meaningful, thus excluding from the dataset those process models that refer to a single organisation. As a result the filtering activity gave us a set of 966 collaboration models, 887 of which can be dealt with by our syntax (i.e., more than 91% of the available collaborations can be analysed by our approach). On this dataset, we successively performed a preliminary transformation step from *.json* (the repository format) to *.bpmn* (the format we manage) and then we ran $\mathcal{S}^3$ to check safeness, soundness and message-relaxed soundness properties.

Experiment Results. Table 1 reports the results of our validation on the models from the BPM Academic Initiative repository. In particular, we cluster the models according to the number of BPMN elements they contain. In the left part of the table we report the number and percentage of models in each class satisfying the properties we are interested in. Concerning safeness, it results that most of the models are safe (on the total around 94%), independently from their size. We can conclude that modelling safe models is a quite common practice, or in any case it is not difficult to respect the property. In relation to soundness, and message-relaxed soundness properties, the situation is rather different. Not surprisingly we notice that the percentage of correct models decreases as the model size increases. On the total only 7% of the models resulted to be sound, while 15% are message-relaxed sound. Therefore, the relaxed form of soundness distinguishes a relevant number of models, the percentage (more than 8%, corresponding to 76 models) seems to justify its relevance in practical contexts. At the same time, it is evident that soundness is not a property easy to satisfy, in particular when collaborations reach a

[3] https://bitbucket.org/proslabteam/s3-validation/src/master/S3.jar.

[4] The repository has been recently dismissed. We have used for our experiments a copy of the whole dataset previously stored on one of our machines. The used models are available in the "Collaboration Models" folder at: https://bitbucket.org/proslabteam/s3-validation/src/master/.

[5] Connectedness evaluates the size of the largest connected sub-graph against the size of the overall model.

Table 1
Fraction of models satisfying the considered properties, and verification time in milliseconds.

Class	*Mod.*	*Safe*	*Sound*	*M.Rel. Sound*	*T.Min*	*T.Max*	*T.Avg.*	*Std.Dev.*
– 9	276	271 (98,19%)	31 (11,23%)	68 (24,64%)	0,025	9,695	0,459	1,059
10–19	354	344 (97,18%)	20 (5,65%)	50 (14,12%)	0,028	4379,046	14,969	234,010
20–29	164	137 (83,54%)	7 (4,27%)	9 (5,49%)	0,110	35628,678	272,262	2845,491
30–39	67	61 (91,04%)	4 (5,97%)	11 (16,42%)	0,224	24682,741	515,556	3066,881
40+	26	24 (92,31%)	0 (0%)	0 (0%)	0,498	20673,422	1910,221	4986,349

certain dimension. The availability of tools for the automatic verification can be considered an added value for the community, and their usage should be fostered.

The right part of Table 1 provides instead some insights on the complexity of the considered models in relation to the time necessary to verify the mentioned properties[6]. As it can be observed, the average in verification time slightly increases with the dimension of the model. We report only one value, since the three properties are verified in a single visit of the resulting LTS model.

A rather high variability with respect to the average time can be observed in relation to the various classes. High values for the standard deviation are indeed not surprising. The time needed for the verification is somehow directly related to the behavioural complexity of a model, that is not strictly correlated to the number of elements in a model. In particular, verification activities are particularly affected by the presence of notation elements leading to interleaved execution, such as parallel or pool elements, that are not always included in models. For example in the class "20–29" there are some models that require an analysis time higher than that of any model in the class "30–39". Nonetheless, it is important to note that in any case the tool was able to provide an answer in reasonable time, also for the most complex models in the dataset. More details on the performance evaluation of the $\mathcal{S}^3$ tool will be discussed later in Section 5.3.

Summing up, answering to the research question RQ1, we can say that the usage of an open and widely used repository confirmed that it is not seldom to find models that violate relevant behavioural properties, also after their release.

5.2. Involvement of practitioners

In this subsection we report the results we get from those experiments that intended to provide possible indications in relation to the research questions RQ2 and RQ3. Hence, the objectives of the experiments are to establish if the proposed approach can actually help designers in getting better models, and if the usage of the supporting tool $\mathcal{S}^3$ is perceived as useful in relation to the modelling of BPMN collaborations.

Experiment Set-Up. In setting up the experiment we involved 26 students enrolled at University of Camerino at the 2nd year of the MSc in Computer Science (Enterprise Software Systems curriculum). MSc students cannot be certainly considered experts in process modelling; however, they cannot be neither considered novice in the discipline: all of them have got a BSc degree in Computer Science and have taken two courses at master level about business process modelling. Thus, the students selected for the experiments can be considered knowledgeable with process modelling practices and quality aspects for BPMN models, even though not really experts. In addition, as suggested by other studies (Gemino and Wand, 2005; Mendling et al., 2010), the involvement of students for the kind of experiments we needed to run is considered rather effective, as students are not biased by prior practical knowledge and experience, that could influence the final results.

To run the modelling experiment, we split the students into two groups of equal cardinality. The modelling activity asked to solve two different exercises, which were conceived to show an increasing level of complexity in BPMN collaboration modelling (see Appendix B for the text of the exercises). The activity had to be finished in two hours, at most. Each member of the first group (referred here as Group A) was asked to solve the modelling exercise only using the *bpmn.io* modelling environment, with no automatic support for quality checking. The members of the second group (referred here as Group B), instead, performed the same activity having the possibility to activate the $\mathcal{S}^3$ functionality, which are integrated in *bpmn.io*, in order to check the possible successive versions of their models.

At the end of the two hours we asked to the students belonging to Group B to fill the questionnaire reported in Appendix C, in order to judge the $\mathcal{S}^3$ tool in relation to its usability and perceived usefulness. The questionnaire has been conceived according to the guidelines provided in Brooke et al. (1996). For each of the considered questions, the student could mark just one box, that would best describe his/her experience with the usage of the $\mathcal{S}^3$. To evaluate the results we got, we successively used the System Usability Scale (SUS), that as for its description "provides a quick and dirty reliable approach for measuring usability" (Bangor et al., 2008).

Experiment Results. After the students ended the modelling activity we checked all the delivered models using $\mathcal{S}^3$. Given that two students (one in Group A and one in Group B) did not complete the second exercise we got 26 models for the first exercise, and 24 for the second one. In Table 2 we report the results of running $\mathcal{S}^3$ on all the models delivered by the students.[7]

The data seems to clearly suggest that a modelling activity supported by $\mathcal{S}^3$ can provide better results, in particular in reference to sound related characterisations. Indeed it is quite evident that students in Group A hardly got sound models, while this is not the case for students in Group B that always got models at least message-relaxed sound. Assuming that there was no relevant differences in the distribution of modelling skills within the two groups, we can argue that the students in Group B could apply an iterative process so to revise their models on the base of the verification results and counterexamples reported by $\mathcal{S}^3$. On the other hand, the data tells us that students do not have much difficulties in deriving safe models, independently from the usage of a supporting tool.

Summing up, we can conclude that, regardless from the scenario to model, the students in Group B outperformed the ones in Group A, in particular in relation to the capability to produce models respecting soundness related properties. This seems to suggest that we can positively answer to the research question RQ2, because designers without a proper support do not easily deliver models respecting relevant properties.

[6] Experiments have been carried out on a machine equipped with a i5-6300U CPU @ 2.40GHz and 16 Gb of RAM.

[7] The models developed by the students are available at the following link https://bitbucket.org/proslabteam/s3-validation/src/master/ in the "Validation with Students" folder, where we have also included an Excel file summarising the results we got.

Table 2
Results from the validation.

Scenario	*Group A*				*Group B*			
	# Mod.	*Safe*	*Sound*	*MR Sound*	*# Mod.*	*Safe*	*Sound*	*MR Sound*
Scenario 1	13	11 (85%)	2 (15%)	1 (8%)	13	11 (85%)	8 (62%)	5 (38%)
Scenario 2	12	12 (100%)	0 (0%)	0 (0%)	12	12 (100%)	3 (25%)	9 (75%)

Table 3
Results from the questionnaires on usability.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Student 1	2	2	4	2	3	1	4	2	5	3
Student 2	3	1	4	1	2	2	4	2	5	2
Student 3	3	1	5	2	4	3	4	1	5	3
Student 4	3	1	5	2	3	2	5	1	4	3
Student 5	3	1	5	2	3	2	4	3	4	3
Student 6	3	1	4	2	3	2	4	3	4	3
Student 7	3	2	4	3	3	2	3	3	5	4
Student 8	4	2	4	3	3	3	4	3	4	2
Student 9	4	1	3	1	2	3	4	2	4	2
Student 10	4	1	4	1	4	2	4	2	3	2
Student 11	3	1	4	1	3	2	3	3	3	3
Student 12	3	3	4	2	3	4	5	2	4	1
Student 13	2	1	4	2	3	3	5	2	5	2

To answer to the research question RQ3, we considered the results of the questionnaire filled by the students in Group B. Table 3 reports the answers provided by the students. The strategy used in the questionnaire alternates questions for which the "positive" answer is somehow inverted. In particular, for questions 1, 3, 5, 7 and 9 the greater the number the better, while for questions 2, 4, 6, 8 and 10 the smaller the number the better. This technique somehow tries to avoid answers provided in a superficial way. Once all administered questionnaires have been filled, the following formula has been computed:

$$\mathcal{S} = \frac{\sum_{i=1}^{N}(\sum_{j=1}^{5}(5 - res_{i,2j}) + \sum_{j=0}^{4}(res_{i,2j+1} - 1))}{N} \times 2.5$$

It provides a way to get an overall score ($\mathcal{S}$) that can be used to globally assess the experiment results. In the formula, N represents the number of returned questionnaires, while $res_{i,j}$ is the response provided to the questionnaire by each participant (i) to each single question (j). The formula at first reconducts all the answers to the scale 0 – 4, where positive answers now always correspond to higher values. Then, for each questionnaire the total sum is computed, getting a number between 0 and 40, and the average over all the questionnaire is derived. Finally, the number is multiplied by 2.5 to get a final score $\mathcal{S}$ in the range 0 – 100. For the experiment we ran, the calculation of the SUS score gives us the value 70.38. According to the proponents of the SUS approach, this is somehow a good result. Indeed, in their experience values for $\mathcal{S}$ greater then 68 relate to a perceived usability somehow better with respect to other used software. Being the involved users somehow experienced with modelling tools, this suggest that overall they got a relatively positive experience in using $\mathcal{S}^3$.

Going more in detail, in Table 4 we report the distribution of the evaluation over the range 0 – 4 for each question. We can observe that the best evaluation was obtained by question 2 ("*I found $\mathcal{S}^3$ unnecessarily complex*") that got an average score of 3.6, while the worst evaluation was reported by question 5 ("*I found that the various functions in $\mathcal{S}^3$ were well integrated*") with an average score of 2. On the positive side, the tool seems easily understandable in its usage, on the negative side it seems required a better integration of the tool with the chosen modelling environment (*bpmn.io* in our case).

5.3. Performance validation

In this subsection we present the experiments we performed to provide an answer to the research question RQ4. To assess the suitability of $\mathcal{S}^3$ in practice, we compare its performances with respect to the performances reported by a tool commonly used with a similar aim, that is the well-known LoLA[8] model checker. The experiment has been inspired by the work reported in Fahland et al. (2009). Among the verification tools considered in that work, we have selected the LoLA model checker, since it is largely used by practitioners and does not exploit GUI or hosting environments that can make hard the retrieval of performance related measures. Anyway, limiting the comparison to LoLa is not an issue for our validation, since our intention here is not to have a complete comparison with different tools, but to get indications on the performance of $\mathcal{S}^3$ with respect to a widely used verification tool. This would possibly permit to conclude that performances related aspects should not be a relevant hurdle for the adoption of $\mathcal{S}^3$ by practitioners.

Experiment Set-Up. For this experiment we restricted the comparison to processes (i.e., single pool models), since the other tool does not consider the verification of correctness properties at collaboration level. Moreover, in setting up the experiment with LoLA we could not run the tool as a batch process, so to feed it with all the BPMN process models included in the BPMN Academic Initiative repository, checking them in just one run. Instead we needed to perform some preprocessing on the *.bpmn* models, including some manual activities (see below). For this reason, we have restricted the experiment to just 20 process models. The selection of these models followed this approach: (i) we performed a massive run of $\mathcal{S}^3$ on the BPMN process models of the BPMN Academic Initiative dataset, in order to identify those models for which $\mathcal{S}^3$ would have reported the worst performance values; (ii) from the ordered list we removed those models that could not be handled by LoLA; (iii) we formed then a first group with 10 models; (iv) the second group, again with 10 models, was instead formed by including those with the greatest number of elements.

The processes were downloaded from BPMN Academic Initiative, and then checked in $\mathcal{S}^3$ using the Java stand-alone version. To perform the verification of the same models in LoLA we had to translate at first the *.bpmn* file of each model into the *.pnml* format, using the export functionalities of the Apromore[9] platform, successively the *.pnml* file has been translated into the *.lola* format using the PN_SUITE[10]. Moreover, before running LoLA, we had to manually annotate the initial marking of each *.lola* file, and to add an edge between the sink place and the source place. Then, the LoLA model checker was launched using a bash-script measuring the execution time needed for the verification. The experiments have been executed, like those illustrated in Section 5.1, on a i5-6300U CPU @ 2.40GHz machine with 16 Gb of RAM; the source files to possibly replicate the experiments are available on GitHub[11]. Each

[8] http://service-technology.org/lola/.
[9] https://apromore.org/.
[10] https://github.com/tamarit/pn_suite.
[11] The models used in this experimentation are available in the "LoLa Comparison Models" folder at https://bitbucket.org/proslabteam/s3-validation/src/master/.

Table 4
Distribution of scores for each question (over the range 0 – 4).

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
0	0	0	0	0	0	0	0	0	0	0
1	2	0	0	0	2	1	0	0	0	1
2	8	1	1	2	9	4	2	5	2	6
3	3	3	9	7	2	7	8	6	6	5
4	0	9	3	4	0	1	3	2	5	1
Avg	2.08	3.62	3.15	3.15	2	2.62	3.08	2.77	3.23	2.46

Table 5
Results from the performance validation in ms (models selected on the basis of the worst performances in $\mathcal{S}^3$).

Models	*LoLA*		$\mathcal{S}^3$	
	Avg.	*St.Dev.*	*Avg.*	*St.Dev.*
1390927551_rev1	25,81	4,40	48850,09	1096,39
899487223_rev3	28,07	24,32	19302,97	722,96
361022685_rev1	Overflow	Overflow	10850,53	795,54
1792465064_rev20	Overflow	Overflow	10211,60	413,54
1962116660_rev2	37,99	9,03	9555,20	555,20
79161580_rev12	23,41	4,98	6722,67	214,53
194092341_rev7	43,96	18,04	5224,30	203,75
1291738678_rev5	25,81	4,40	3765,05	225,93
1269449021_rev7	39,33	45,56	2986,90	241,10
1576721033_rev9	20,78	8,53	1309,45	128,79

model has been verified 10 times, with each tool, and the average time has been used and reported for the comparison.

Experiment Results.

Table 5 reports the results of our experiments in relation to the worst performance for $\mathcal{S}^3$, while Table 6 shows the results of the experiments for the models with the greatest number of elements.

The models in Table 5 that are somehow difficult to verify for $\mathcal{S}^3$ appeared to be much simpler to handle by LoLa. We observe that $\mathcal{S}^3$ suffers when many activities are executed in parallel, and indeed all models in the list include parallel behaviours. However, it is also worth mentioning that for two of the considered models LoLA, differently from $\mathcal{S}^3$, was not able to conclude the verification due to memory overflow.

From the results reported in Table 6, instead, we can observe that $\mathcal{S}^3$ seems to perform faster than LoLA, which in 4 cases was even not able to complete the analysis. Taking a look to these models in more detail, the impression is that LoLA could suffer more than $\mathcal{S}^3$ from the presence of loops.

From the data we got so far, it seems that verification strategies for the two tools are rather different, and there is not an evident correlation in observed performances. A final information worthy to be reported concerns the fact that the highest verification time reported by $\mathcal{S}^3$ on the BPM Academic Initiative dataset is about 48 seconds. In summary, even if technical hurdles made difficult to provide a clear definite answer to the research question RQ4, the performance results seem to suggest that $\mathcal{S}^3$ is able to complete the verification in reasonable amount of time with respect to LoLA.

5.4. Threats to validity

The evidences we got from the experiments we ran have to be considered also in light of some threats that could possibly affect the reported results. We include here a set of factors that could have impacted our experiments. The discussion follows the list of research questions introduced at the beginning of this section.

RQ1. The percentages reported in Table 1 clearly show that designers have often released on the BPMN Academic Initiative repository models that do not satisfy relevant behavioural properties. Nevertheless, the dataset does not provide much information on the models source and the objectives of designers. In particular, given a model, it is not possible to know the experience of its designer, and if the model was developed to represent a real process (e.g., to possibly intervene on it or to implement a supporting software) or if, instead, it has been developed in an educational context to show typical modelling issues. Given the reported low percentages, we could suppose that some of the models have been developed outside of practical application context. Clearly, this could have affected the results, and then our conclusions. However, the really low percentages reported let us think that in any case there is a fraction of models released by professionals that do not satisfy important behavioural properties. As already mentioned in Section 5.1, the need of datasets on which to experiment new proposals is one of the main issues that the BPM community is currently trying to face.

RQ2. The experiments we carried out with "junior" practitioners made possible to conclude that $\mathcal{S}^3$ effectively supports designers in getting higher quality models. Also in this case the percentages are quite clear. However, the two groups were quite small and, even though the students formally are taking the same curriculum in relation to the business process modelling related experiences, it could be possible that the group B included the best students. In such a case, the result would not be directly related to the tool, but on the abilities of the students. At the same time, the lack of an ample experience in real modelling contexts could lead to results that are not representative of real modelling contexts. Indeed, experienced practitioners could compensate the lack of a supporting verification tool with their capability of producing high quality models. To avoid these effects, the experiment would have involved more experienced practitioners. Nonetheless, it is well known that this comes with many other problems, not last the need of a possibly relevant budget.

RQ3. In this case the results are not clearly definitive, since we got a value slightly greater than 68, that is considered the threshold over which the feedback can be considered positive. On the one hand, this could be justified by the fact that $\mathcal{S}^3$ is still a prototype. Even though no bug emerged during the experiment, the perception of a software still under development could give an unsatisfactory feeling to the user. On the other hand, it is possible that the students, even though the questionnaires were anonymous and that they were explicitly asked to provide reliable answers, avoided to provide harsh comments, perhaps thinking that we could retrieve their identity, or for a form of kindness towards their teachers. Anyway, we intend to carry out further validations with successive versions of the tool.

RQ4. Considerations similar to those made for RQ1 can be applied to this case. The BPMN Academic Initiative repository has been chosen as it includes many models (thousands), and it permits to use models defined by others. However, it is difficult to select subsets of models according to specific quality dimensions, or defined in relation to concrete working contexts. Selected models could be not really relevant, and the results for the checked properties could not be fully representative of the all possible properties that a designer could be interested in checking. In addition, to derive the *.lola* format we had to use two tools in sequence

Table 6
Results from the performance validation in ms (models selected on the basis of their size).

Models	*Number of*	*LoLA*		$\mathcal{S}^3$	
	Elements	*Avg.*	*St.Dev*	*Avg.*	*St.Dev.*
2025710215_rev4	86	67,40	6,23	11,66	4,94
536365423_rev11	85	Overflow	Overflow	7,84	1,08
1792465064_rev20	74	Overflow	Overflow	9710,70	396,24
1396260552_rev10	67	Overflow	Overflow	1,61	1,30
1020811925_rev5	64	68,75	77,16	13,38	10,90
94543997_rev1	64	47,49	10,06	3,68	1,41
652975530_rev17	62	Overflow	Overflow	4,75	2,33
1869279615_rev19	59	55,87	8,66	0,96	0,53
412368141_rev2	57	44,42	20,33	4,41	1,82
1508335739_rev1	52	37,92	6,22	2,87	1,37

(Apromore and PN_TOOLS), and it is possible that the implemented translations could have an impact on the LoLA verification activities. It is worth mentioning that $\mathcal{S}^3$ is still a prototype, which can be subject to many optimisations, while LoLA is a mature product used since many years by a large community.

In summary, the experiments we have performed gave us useful indications with respect to the selected research questions. However, further confirmations will be searched extending the validation experience. In particular, experiments using models coming from real and heterogeneous contexts, and feedbacks by senior practitioners could reinforce the initial indications we got.

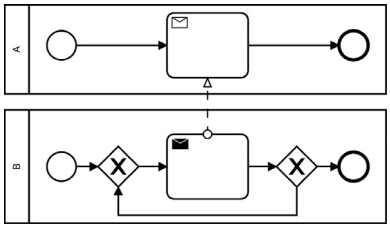


Fig. 11. Example of a safe and message-relaxed sound BPMN collaboration model.

6. Related works

Much effort has been devoted to the formalisation and verification of business processes (Morimoto, 2008; Groefsema and Bucur, 2013; Fellman and Zasada, 2016), but to the best of our knowledge there is not a research work satisfying all the following requirements: (i) providing a direct formalisation taking into account message exchange and sub-process behaviour, as well their impact on collaborations; (ii) providing a relaxed variant of soundness to distinguish issues related to the process internal behaviour from those related to message exchange; (iii) providing a user-friendly tool integrating modelling activity with the verification of safeness and soundness properties.

BPMN Formalisation. Different semantics have been proposed in the literature and according to their definitions we can distinguish formalisations given directly on the BPMN syntax and the others that rely on mappings into other languages.

Among the direct formalisations, in Van Gorp and Dijkman (2013) the authors propose a formalisation of the BPMN 2.0 execution semantics in terms of graph transformation rules. The formalisation is documented using visual rules that update a BPMN model in-place. With respect to our work, the used formalisation techniques are different, since we provide an operational semantics in terms of LTS, which allows us to apply verification techniques well-established for this underlying model, such as model checking.

The definition of an operational semantics gives us the possibility to be tool independent rather than be constrained to tools specific for graph transformation rules. This is confirmed by the same authors that for using their BPMN formalization in the compliance verification between global and local process models need a further transformation (Kwantes et al., 2015). Another direct semantics was defined in El-Saber and Boronat (2014), where the authors propose a formal specification of well-formed BPMN processes in rewriting logic using Maude, with a focus on data-based decision gateways and data objects semantics. In Börger and Thalheim (2008) the authors present a formal framework relying on rule-based meta language and its application to generic control structures of business processes. In Kossak et al. (2014) the authors in order to rigorously specify the semantics of BPMN, propose a direct formalisation in abstract state machines. Finally, in Christiansen et al. (2011) the authors gives the semantics reducing the syntax to a subset of BPMN 2.0 elements containing just the inclusive and exclusive gateways, and the start and stop events. Summing up, all the cited papers focus on a direct semantics, as well as our approach; however, none of them is able to reason on collaboration including at the same time sub-processes, messages and their interplay.

Considering the formalisation given via translations, we can find in the literature different target languages, such as Petri Nets (Dijkman et al., 2008; Koniewski et al., 2006; Ramadan et al., 2011; Awad et al., 2010; 2008), process calculi (Wong and Gibbons, 2011; 2008; Arbab et al., 2008; Prandi et al., 2008; Puhlmann and Weske, 2006; Puhlmann, 2007; Puhlmann and Weske, 2005), EPC (Mendling, 2007), ECATNets (Kheldoun et al., 2015), and YAWL (Ye and Song, 2010; Decker et al., 2008). All these approaches suffer from issues related to encodings. In fact, the semantics given by translation is based on the low-level details of the encoding without consider the actual features and constructs of BPMN. This may make the verification results inaccurate, since translations usually rely on assumptions and language abstractions. More in details, considering the mapping from BPMN to Petri Nets, the one proposed by Dijkman et al. in Dijkman et al. (2008) is probably the most relevant contribution. It enables the use of standard tools for process analysis, in order to check absence of deadlock and proper completion of BPMN models. However, the approach proposed by Dijkman et al. is based on the version 1.1 of BPMN and, as the authors stated, it suffers from deficiencies that impact on the proposed formalisation. Moreover, even if the mapping deals with messages, differently from our approach it does not properly consider multiple organisation scenarios making clear who are the participants involved in the exchange of messages, and it works only under the assumption that sub-processes are safe. In addition, the notion of safeness differs when it is given directly on BPMN collaborations, with respect to the notion of safeness for Petri Nets applied as it is to translations of BPMN collaborations according to the mapping in Dijkman et al. (2008). In fact, safeness of a BPMN collaboration only refers to the tokens on the sequence edges of

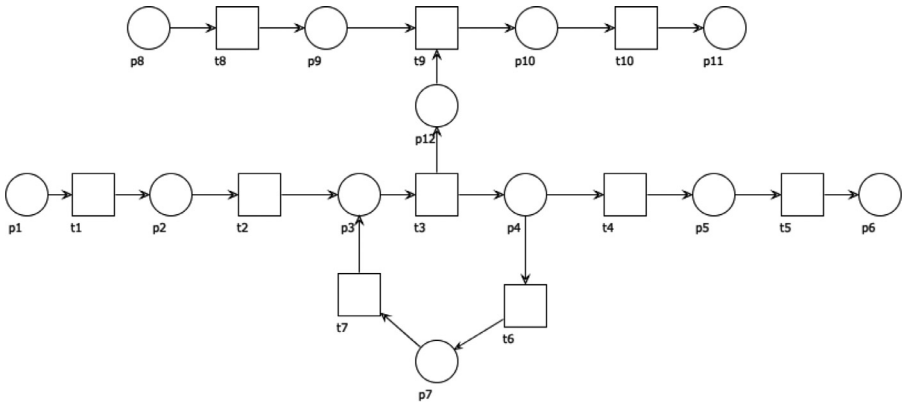


Fig. 12. Petri Net derived from the model in Fig. 11.

the involved processes, while in its Petri Nets translation refers to both message and sequence edges. Indeed, such distinction is not considered in the translations, because a message is rendered as a (standard) token in a place. Hence, a safe BPMN collaboration may be considered unsafe by relying on the Petri Nets notion. Summing up, such approaches do not deal with all the BPMN specification and its peculiarities. In principle, we do not envisage major hurdles in extending the translations from BPMN to Petri Nets available in the literature in order to achieve results similar to ours. However, we have preferred to develop a direct semantics because extending available translations may result in generation of convoluted and large Petri Nets, thus undermining the understanding of the formal meaning of the BPMN execution semantics, and the verification of BPMN collaborations.

Also process calculi have been considered as formalisation languages for BPMN, typically enabling verification by means of model checkers. In particular, in Wong and Gibbons (2011) a translation into a CSP-like language for a subset of well-formed BPMN process diagrams including sub-processes and messages is proposed. However, differently from this paper, the verification they propose is the consistency checking, performed by using the FDR tool. In Puhlmann and Weske (2005) the authors introduce a collection of workflow patterns formalisations based on the π-calculus. The formalisations can be used as a foundation for pattern-based workflow execution, reasoning, and simulation. The common approaches for the automatic verification of these formalisations are based on model checking (Koehler et al., 2002; Klai et al., 2009) with the limitation of checking, differently from our approach, only reachability properties. Summing up, all the verifications performed by means of these approaches suffer from expressivity, usability and performance issues related to the used logics and model checkers. We overcome such issues as our verification is ad-hoc, rather than a general purpose verification framework. This is particular useful on complex properties such as soundness.

BPMN Correctness Properties. Concerning verification, different properties have been defined in the context of Petri Nets. The most known one is the structural soundness, defined in Weske,Van Der Aalst et al. (2011), that guarantees the absence of deadlock and livelock. Other relaxed notions of soundness is defined allowing the occurrence of deadlocks and livelocks. The relaxed sound was proposed in Dehnert and Rittgen (2001), admitting at least one proper execution so considering deadlocks and livelocks as non-invalidating behaviours. Weak soundness in Martens (2005) disallows deadlocks, but it allows certain parts of the process not to participate in any process instance (i.e., it admits dead activities). For critical control flow patterns, where the process model can be neither sound nor weak sound, lazy soundness has been proposed as a new soundness criterion by the authors in Puhlmann and Weske (2006).

The works mentioned above focus mainly on the verification of correctness properties related to the control flow of the business process, without considering communication aspects. None of the available encodings of BPMN in Petri Nets, or similar formalisms, permit to distinguish sequence and message edges (as discussed above). Correspondingly analysis activities based on such encodings are not able to differentiate issues concerning the control flow from those concerning the message flow.

To clarify this point, we resort to an example. Let us consider the collaboration model in Fig. 11, which is safe and message-relaxed sound with respect to our notions of the properties. Using the mapping from BPMN to Petri Nets proposed by Dijkman et al. (2008), the resulting Petri Net is the one shown in Fig. 12. Checking the safeness and soundness properties of this Petri Net (using, e.g., the well-known tool WoPeD[12]), it instead results to be unsafe and unsound. In fact, it is unsafe because more than one token can mark the place $p12$, produced by the translation of the message edge, and it is unsound because when the collaboration completes (places $p6$ and $p11$ are marked) another token is still around in the net.

In our case the definition of a direct formalisation for the BPMN notation permits to provide definitions for relevant properties, such as soundness and safeness relying on the relevant characteristics of BPMN models. In particular, according to our view the presence of multiple enqueued messages should not be considered a sufficient reason for considering the model completely unsound. The verification of message-relaxed soundness permits to distinguish such situations.

BPMN Verification Tools. Let us consider the literature about the proposed tools to support the verification of BPMN models. GrGen.NET (Jakumeit et al., 2010) is a graph rewrite tool enabling the development of a BPMN models. The authors provide two formalisations, one in YAWL for the execution and one in Petri Nets, which allow BPMN models to be formally analysed. The authors in Kheldoun et al. (2017) propose a verification approach for BPMN models based mainly on meta-modeling and model transformations. The RECATNets formalism is used as intermediate language for having a rewriting logic description, together with LTL formulas, as input for the Maude model checker. BPA tool (Eid-Sabbagh et al., 2013) is a framework used for structuring and managing process collections. The authors were able to model and analyse the correctness of a business process, by transforming it into open nets, translate the correctness criteria into CTL formulae, and

[12] https://woped.dhbw-karlsruhe.de/.

model check those formulae using LoLA. In Huai et al. (2010) the authors propose an approach that takes in input a BPMN file and apply a transformation in Petri Net modules. The properties verification proposed (dead tasks, deadlocks and loops) were checked on the generated reachability graph. The BPMN2YAWL tool described in Decker et al. (2008) takes a BPMN diagram as input and produces two files: the XMI representation of the model and the layout information. The XMI file can be successively transformed into an input file for YAWL that allows to execute workflow systems.

Summing up, the majority of the tools propose BPMN translations and verification of standard properties, like deadlock and livelock, via model checking. Instead, $\mathcal{S}^3$ supports soundness checking natively. In addition, from the usability perspective, our implementation allows the designer to model and simultaneously check soundness and safeness properties with the same tool during the design phase, getting feedbacks in case of errors. Finally, from the validation point of view, the mentioned tools present simple examples as proof of concept, without test them in an extensive manner. $\mathcal{S}^3$ instead was extensively validated on models coming from the BPM Academic Initiative repository.

7. Concluding remarks and future work

In this paper we propose a theory, and a related supporting tool, for checking safeness and soundness of BPMN collaboration models. We started defining the formal semantics for a subset of BPMN collaboration elements, including in particular sub-processes and the elements enabling exchange of messages.

Our direct semantics, instead, aims at formalising BPMN features as close as possible to their definition in the standard specification, without any bias from the use of another formalism. A faithful semantics, of course, ensures a more effective verification of model properties. On top of the defined semantics we formally characterised the notions of safeness and soundness for BPMN models, also including the possibility of considering or not the impact of pending messages on soundness. This has a practical relevance, as our validation as shown that around 15% of the analysed models are unsound but message-relaxed sound. The semantics and the property verification have been implemented in the $\mathcal{S}^3$ tool available as an on-line service. This has been integrated with the Camunda modeller, thus providing a fully-integrated modelling and verification environment. The usage of the tool and its real effectiveness have been checked by means of an open repository, that permitted to run a validation using 887 models.

In the future, we plan to extend our framework to overcome some current limitations of the proposed approach. The verification framework abstracts from data values. This is motivated by the fact that our verification aims at exhaustively analysing all possible executions of a given model. Anyway, an extension of our framework including the data perspective would allow to deal with fine-grained data-aware specifications, in order to identify those specific issues related to data, as e.g. dead activities, malformed messages, wrongly delivered messages, etc. We also aim at extending our approach to a wider set of BPMN elements, such as exceptions, multi-instance tasks and pools and timed events. This will allow us to have a wider coverage of the BPMN standard, and will impact both our formal treatment and the $\mathcal{S}^3$ tool. We intend to extend our validation activity, as already discussed at the end of Section 5.4, in order to limit the discussed threats to validity. Finally, concerning verification, we plan to enable the checking of a wider set of correctness properties.

Appendix A. Proof

We report here the proofs of results in Section 4.

Theorem 1 (Safeness Correspondence) *Let C be a collaboration and $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ its LTS, then C is safe if and only if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is safe.*

Proof. We prove below the *if* and the *only if* parts of the theorem.

- (*if* part) In this case we have to show that if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is safe then C is safe. The proof proceeds by contradiction. Suppose that C is unsafe. By Def. 2, given σ and δ such that $isInit(\langle C, \sigma, \delta \rangle)$, there exist σ' and δ' such that $\langle C, \sigma, \delta \rangle \twoheadrightarrow \langle \sigma', \delta' \rangle$ and $\exists \mathsf{e} \in \mathbb{E}.\sigma'(\mathsf{e}) > 1$. Hence, by Def. 5, $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is unsafe, which is a contradiction.
- (*only if* part) In this case we have to show that if C is safe then $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is safe. The proof proceeds by contradiction. Suppose that $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is unsafe. This means, by Def. 5, that there exists $\langle C, \sigma, \delta \rangle \in \mathcal{C}$ such that $\exists \mathsf{e} \in \mathbb{E} \,.\, \sigma(\mathsf{e}) > 1$. Hence, by Def. 2, C is unsafe, which is a contradiction. □

Theorem 2 (Soundness Correspondence) *Let C be a collaboration and $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ its LTS, then C is sound if and only if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is sound.*

Proof. We prove below the *if* and the *only if* parts of the theorem.

- (*if* part) In this case we have to show that if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is sound then C is sound.
  By definition, $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is sound iff $\exists! \langle C, \sigma'', \delta'' \rangle \in \mathcal{C}$ such that: (i) $\langle C, \sigma'', \delta'' \rangle \cdot \xrightarrow{l}$ and (ii) $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, incoming(C, \sigma'', \delta''))$ and (iii) $incoming(C, \sigma'', \delta'') = endIn(C)$ and (iv) $\forall \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) = 0$ and (v) $\forall \mathsf{m} \in \mathbb{M}.\delta''(\mathsf{m}) = 0$
  By contradiction:
  - Let suppose that $\nexists \langle C, \sigma'', \delta'' \rangle$ in which *i, ii, iii, iv, v* holds together. This means that at least one of those conditions is violated. Consequently, we have to consider the following cases.
    - $\neg(\langle C, \sigma'', \delta'' \rangle \cdot \xrightarrow{l})$
      This implies that $\langle C, \sigma'', \delta'' \rangle \xrightarrow{l} \langle \sigma''', \delta''' \rangle$ hence either $l = \mathsf{e}$ then $\sigma''(\mathsf{e}) > 0$, $l = E$ then $\forall \mathsf{e} \in E.\sigma''(\mathsf{e}) > 0$, or $l = !\mathsf{m}$ or $l = ?\mathsf{m}$ then $\delta''(\mathsf{m}) > 0$. In each case, the collaboration is *unsound*, which is a contradiction.
    - $\neg(isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, endIn(C)))$
      This implies that $\exists l \in endIn(C)$, $\langle C, \sigma, \delta \rangle$, $\langle C, \sigma', \delta' \rangle$ such that $\langle C, \sigma, \delta \rangle \overset{l}{\Longrightarrow} \langle C, \sigma', \delta' \rangle \overset{l}{\Longrightarrow} \langle C, \sigma'', \delta'' \rangle$. Thus, there exist an end event with l as incoming label that has been reached more than once. Consequently, $\exists \mathsf{e}_{cmp} \in marked(\sigma'', end(C)).\sigma''(\mathsf{e}_{cmp}) > 1$, the collaboration is *unsound* which is a contradiction.
    - $\neg(incoming(C, \sigma'', \delta'') = endIn(C))$
      This implies that $incoming(C, \sigma'', \delta'') \neq endIn(C)$. In the case that $\exists l \in incoming(C, \sigma'', \delta'')$, with $l \notin endIn(C)$, we have that the state $\langle \sigma'', \delta'' \rangle$ can be reached by consuming the label l that is not an incoming label of an end event. Consequently, the state $\langle \sigma'', \delta'' \rangle$ is either a state of deadlock or a non-final state, thus the collaboration is *unsound*, which is a contradiction. The case that $\exists l \in endIn(C)$, with $l \notin incoming(C, \sigma'', \delta'')$, is similar.
    - $\neg(\forall \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) = 0)$
      This implies that there exists an edge $\mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) > 0$. Consequently, the collaboration is *unsound* which is a contradiction.
    - $\neg(\forall \mathsf{m} \in \mathbb{M}.\delta''(\mathsf{m}) = 0)$
      This implies that there exists a message $\mathsf{m} \in \mathbb{M}.\delta''(\mathsf{m}) > 0$. Consequently, the collaboration is *unsound* which is a contradiction.
  - Let suppose that there exist $\langle C, \sigma'', \delta'' \rangle$ and $\langle C, \sigma''', \delta''' \rangle$ in which *i, ii, iii, iv, v* hold together, with $\exists \langle C, \sigma'', \delta'' \rangle \neq \langle C,$

σ''', $\delta'''\rangle$. This means that either $\sigma'' \neq \sigma'''$ or $\delta'' \neq \delta'''$. This contradicts (iv) and (v), thus the collaboration is *unsound* which is a contradiction.

- (*only if* part) In this case we have to show that if C is sound then $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is sound.
  By definition, let $\langle C, \sigma, \delta \rangle$ be the collaboration configuration such that $isInit(\langle C, \sigma, \delta \rangle)$ then $\forall \sigma'$ and δ' such that $\langle C, \sigma, \delta \rangle \rightarrow^* \langle \sigma', \delta' \rangle$, $\exists \sigma''$ and δ'' such that $\langle C, \sigma', \delta' \rangle \rightarrow^* \langle \sigma'', \delta'' \rangle$, *(i)* $\forall \mathsf{e}_{cmp} \in marked(\sigma'', end(C)).\sigma''(\mathsf{e}_{cmp}) = 1$, *(ii)* $\forall \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) = 0$ and *(iii)* $\forall \mathsf{m} \in \mathbb{M}.\delta''(\mathsf{m}) = 0$.
  By contradiction:
  Let suppose that $\nexists \langle C, \sigma'', \delta'' \rangle$ in which *i, ii, iii* hold together. This means that $\forall \langle C, \sigma'', \delta'' \rangle$ at least one of the following condition is violated:
  - $\exists \mathsf{e}_{cmp} \in marked(\sigma'', end(C)).\sigma''(\mathsf{e}_{cmp}) > 1$. This means that there exists $\mathsf{end}(l, \mathsf{e}_{cmp})$ and $\langle C, \sigma_1, \delta_1 \rangle, \langle C, \sigma_2, \delta_2 \rangle, \langle C, \sigma_3, \delta_3 \rangle \in \mathcal{C}$ such that $\langle C, \sigma_1, \delta_1 \rangle \stackrel{l}{\Longrightarrow} \langle C, \sigma_2, \delta_2 \rangle \stackrel{l}{\Longrightarrow} \langle C, \sigma_3, \delta_3 \rangle$. This violate the $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, incoming(C, \sigma'', \delta''))$ making the LTS *unsound*, which is a contradiction.
  - $\exists \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) \neq 0$. Hence, there exists an element of the collaboration with a token in the incoming sequence flow, so that $\forall \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) = 0$ is *false*. Consequently, the LTS is *unsound* which is a contradiction.
  - $\exists \mathsf{m} \in \mathbb{M}.\delta''(\mathsf{m}) \neq 0$. Hence, there exists a message of the collaboration pending, so that $\forall\ \mathsf{m} \in \mathbb{M}.\delta(\mathsf{m}) = 0$ is *false*. Consequently, the LTS is *unsound*, which is a contradiction. □

Theorem 3 (Message-Relaxed Soundness Correspondence) *Let C be a collaboration and $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ its LTS, then C is message-relaxed sound if and only if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is message-relaxed sound.*

Proof. We prove below the *if* and the *only if* parts of the theorem.

- (*if* part) In this case we have to show that if $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is message-relaxed sound then C is message-relaxed sound.
  By definition, $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is message-relaxed sound iff $\forall \langle C, \sigma'', \delta'' \rangle \in \mathcal{C}$ such that: (i) $\langle C, \sigma'', \delta'' \rangle \cdot \stackrel{l}{\rightarrow}$ and (ii) $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, incoming(C, \sigma'', \delta''))$ and (iii) $incoming(C, \sigma'', \delta'') = endIn(C)$ and (iv) $\forall\ \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) = 0$.
  By contradiction, let C be message-relaxed unsound, this means $\exists \langle C, \sigma'', \delta'' \rangle$ where one of the following condition is violated:
  - $\neg(\langle C, \sigma'', \delta'' \rangle \cdot \stackrel{l}{\rightarrow})$
    This implies that $\langle C, \sigma'', \delta'' \rangle \stackrel{l}{\rightarrow} \langle \sigma''', \delta''' \rangle$ hence either $l = \mathsf{e}$ then $\sigma''(\mathsf{e}) > 0$, $l = E$ then $\forall \mathsf{e} \in E.\sigma''(\mathsf{e}) > 0$, or $l = !\mathsf{m}$ or $l = ?\mathsf{m}$ then $\delta''(\mathsf{m}) > 0$. In each case, the collaboration is *message-relaxed unsound*, which is a contradiction.
  - $\neg(isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, endIn(C)))$
    This implies that $\exists l \in endIn(C)$, $\langle C, \sigma, \delta \rangle$, $\langle C, \sigma', \delta' \rangle$ such that $\langle C, \sigma, \delta \rangle \stackrel{l}{\Longrightarrow} \langle C, \sigma', \delta' \rangle \stackrel{l}{\Longrightarrow} \langle C, \sigma'', \delta'' \rangle$. Thus, there exists an end event with l as incoming label that has been reached more than once. Consequently, $\exists \mathsf{e}_{cmp} \in marked(\sigma'', end(C)).\sigma''(\mathsf{e}_{cmp}) > 1$, the collaboration is *message-relaxed unsound*, which is a contradiction.
  - $\neg(incoming(C, \sigma'', \delta'') = endIn(C))$
    This implies that $incoming(C, \sigma'', \delta'') \neq endIn(C)$. In the case that $\exists l \in incoming(C, \sigma'', \delta'')$, with $l \notin endIn(C)$, we have that the state $\langle \sigma'', \delta'' \rangle$ can be reached by consuming the label l that is not an incoming label of an end event. Consequently, the state $\langle \sigma'', \delta'' \rangle$ is either a state of deadlock or a non-final state, thus the collaboration is *message-relaxed unsound* which is a contradiction. The case that $\exists l \in endIn(C)$, with $l \notin incoming(C, \sigma'', \delta'')$, is similar.
  - $\neg(\forall\ \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) = 0)$
    This implies that there exists an edge $\mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) > 0$. Consequently, the collaboration is *message-relaxed unsound* which is a contradiction.
- (*only if* part) In this case we have to show that if C is message-relaxed sound then $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ is message-relaxed sound.
  By definition, let $\langle C, \sigma, \delta \rangle$ be the collaboration configuration such that $isInit(\langle C, \sigma, \delta \rangle)$ then $\forall \sigma'$ and δ' such that $\langle C, \sigma, \delta \rangle \rightarrow^* \langle \sigma', \delta' \rangle$ there must be σ'' and δ'' such that $\langle C, \sigma', \delta' \rangle \rightarrow^* \langle \sigma'', \delta'' \rangle$,
  (i) $\forall \mathsf{e}_{cmp} \in marked(\sigma'', end(C)).\sigma''(\mathsf{e}_{cmp}) = 1$, *(ii)* $\forall \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) = 0$.
  Let suppose that $\nexists \langle C, \sigma'', \delta'' \rangle$ in which *i* and *ii* hold together. This means that $\forall \langle C, \sigma'', \delta'' \rangle$ at least one of the following condition is violated:
  - $\exists \mathsf{e}_{cmp} \in marked(\sigma'', end(C)).\sigma''(\mathsf{e}_{cmp}) > 1$. This means that there exists $\mathsf{end}(l, \mathsf{e}_{cmp})$ and $\langle C, \sigma_1, \delta_1 \rangle, \langle C, \sigma_2, \delta_2 \rangle, \langle C, \sigma_3, \delta_3 \rangle \in \mathcal{C}$ such that $\langle C, \sigma_1, \delta_1 \rangle \stackrel{l}{\Longrightarrow} \langle C, \sigma_2, \delta_2 \rangle \stackrel{l}{\Longrightarrow} \langle C, \sigma_3, \delta_3 \rangle$. This violate the $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, incoming(C, \sigma'', \delta''))$ making the LTS *message-relaxed unsound*, which is a contradiction.
  - $\exists \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) \neq 0$. Hence, there exists an element of the collaboration with a token in the incoming sequence flow, so that $\forall \mathsf{e} \in \mathbb{E} \backslash end(C).\sigma''(\mathsf{e}) = 0$ is *false*. Consequently, the LTS is *message-relaxed unsound*, which is a contradiction. □

Appendix B. Validation

We report here the exercises assigned to students. Results of the validation are discussed in Section 5.2.

Scenario 1

A company produces and assemblies motorbike components for different brands. Whenever the company receives an order from the customer, it analyses the request and activates the production process. Such a process is composed of three parallel activities referring to the production of three main components of the motorbike: the hydraulic system, the electrical circuit, and the engine. As soon as any of these components is produced, it is immediately assembled with the rest of the bike. Notice for security reason only one type of component can be assembled at the same time. When the assembly is fully completed, the bike is ready for the quality check. The quality check is performed by an autonomous participant acting as an external third party. The quality check procedure consists in performing a quality test and in the preparation of the resulting report. The report is sent to the company that verifies the result. Then, the company notifies the order completion to the customer and delivers to him the bike. Before receiving the bike, the customer can possibly receive and check the order completion notification.

Scenario 2

Considering the company of the previous scenario, the production is divided into two departments that are geographically located in two different sheds, one for the assembling and one for the packaging. The two production departments do not communicate with each other but interact with the Quality Office. The Quality Office is devoted to check the quality of the assembled and packaged products. The collaboration starts with the Assembly and the Packaging departments that perform their processes independently. The Assembly department initiates its process by collecting all the items required for the product, and then moves to the assembly activities. This phase consists in the assembly of mechanical and electrical items. The mechanical items are mandatory, while the electrical ones are optional. Then a serial number is assigned to the component and it is registered in the IT system. Finally, the product is sent to the Quality Office for check-

ing conformance with the standards. Parallelly, the Packaging department process starts. The assembled products are covered using one among three different materials: paper, cardboard or plastic. Completed the packaging, a label is applied on each pack. Then the packs pass under an X-Ray scanner, that can reveal potential damages. In case of damages the product is sent to the Quality Office for further inspections, then the process completes. The Quality Office process starts with the login in the IT system. Then the Quality Office waits for receiving a support request coming from either the Assembly department or the Packaging department. Once a request is received, the Quality Office starts to write the evaluation report and it will update the product state in the IT system. As last activity, the quality office will send it back the report to the corresponding department.

Appendix C. Questionnaire

We report here the System Usability Scale questionnaire administered to students. The results of the validation are discussed in Section 5.2.

Question 1 - I think that I would like to use $\mathcal{S}^3$ frequently.

Strongly disagree				Strongly agree
1	2	3	4	5

Question 2 -
I found $\mathcal{S}^3$ unnecessarily complex.

Strongly disagree				Strongly agree
1	2	3	4	5

Question 3 -
I thought $\mathcal{S}^3$ was easy to use.

Strongly disagree				Strongly agree
1	2	3	4	5

Question 4 -
I think that I would need assistance to be able to use $\mathcal{S}^3$.

Strongly disagree				Strongly agree
1	2	3	4	5

Question 5 -
I found the various functions in $\mathcal{S}^3$ were well integrated.

Strongly disagree				Strongly agree
1	2	3	4	5

Question 6 -
I thought there was too much inconsistency in $\mathcal{S}^3$.

Strongly disagree				Strongly agree
1	2	3	4	5

Question 7 - I would imagine that most people would learn to use $\mathcal{S}^3$ very quickly.

Strongly disagree				Strongly agree
1	2	3	4	5

Question 8 -
I found $\mathcal{S}^3$ very cumbersome/awkward to use.

Strongly disagree				Strongly agree
1	2	3	4	5

Question 9 -
I felt very confident using $\mathcal{S}^3$.

Strongly disagree				Strongly agree
1	2	3	4	5

Question 10 -
I needed to learn many things before I could get going with $\mathcal{S}^3$.

Strongly disagree				Strongly agree
1	2	3	4	5

References

Van der Aalst, W., 1995. A class of petri net for modeling and analyzing business processes. Comput. Sci. Rep. 95 (26), 1–25.

van der Aalst, W.M.P., 2000. Workflow verification: finding control-flow errors using petri-net-based techniques. In: Business Process Management, Models, Techniques, and Empirical Studies. In: LNCS, 1806. Springer, pp. 161–183. doi:10.1007/3-540-45594-9_11.

Arbab, F., Kokash, N., Meng, S., 2008. Towards using reo for compliance aware business process modeling. In: Leveraging Applications of Formal Methods, Verification and Validation. In: CCIS. Springer, pp. 108–123.

Awad, A., Decker, G., Lohmann, N., 2010. Diagnosing and repairing data anomalies in process models. In: Business Process Management Workshops. In: LNBIP, Vol. 43. Springer, pp. 5–16.

Awad, A., Decker, G., Weske, M., 2008. Efficient compliance checking using bpmn-q and temporal logic. In: Business Process Management. In: LNCS, 5240. Springer, pp. 326–341.

Bangor, A., Kortum, P.T., Miller, J.T., 2008. An empirical evaluation of the system usability scale. Int. J. Hum.–Comput. Interact. 24 (6), 574–594.

Börger, E., Thalheim, B., 2008. A method for verifiable and validatable business process modeling. In: Advances in Software Engineering. In: LNCS, 5316. Springer, pp. 59–115.

Brooke, J., et al., 1996. Sus-a quick and dirty usability scale. Usabil. Evaluat. Ind. 189 (194), 4–7.

Christiansen, D.R., Carbone, M., Hildebrandt, T., et al., 2011. Formal semantics and implementation of BPMN 2.0 inclusive gateways. In: Web Services and Formal Methods. In: LNCS, 6551. Springer, pp. 146–160.

Corradini, F., Ferrari, A., Fornari, F., Gnesi, S., Polini, A., Re, B., Spagnolo, G., 2018. A guidelines framework for understandable BPMN models. Data Knowl. Eng. 113, 129–154.

Decker, G., Dijkman, R., Dumas, M., García-Bañuelos, L., 2008. Transforming BPMN diagrams into YAWL nets. In: International Conference on Business Process Management. In: LNCS, 5240. Springer, pp. 386–389.

Dehnert, J., Rittgen, P., 2001. Relaxed soundness of business processes. In: International Conference on Advanced Information Systems Engineering. In: LNCS, 2068. Springer, pp. 157–170.

Dehnert, J., Zimmermann, A., 2005. On the suitability of correctness criteria for business process models. In: BPM. In: LNCS, 3649. Springer, pp. 386–391. http://link.springer.com/chapter/10.1007/11538394_28

Dijkman, R.M., Dumas, M., Ouyang, C., 2008. Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. 50 (12), 1281–1294.

Dumas, M., La Rosa, M., Mendling, J., Mäesalu, R., Reijers, H.A., Semenenko, N., et al., 2012. Understanding business process models: the costs and benefits of structuredness. In: Advanced Information Systems Engineering. In: LNCS, 7328. Springer, pp. 31–46. http://link.springer.com/chapter/10.1007/978-3-642-31095-9_3

Eid-Sabbagh, R.-H., Hewelt, M., Weske, M., 2013. A tool for business process architecture analysis. In: International Conference on Service-Oriented Computing. In: LNCS, 8274. Springer, pp. 688–691.

El-Saber, N., Boronat, A., 2014. BPMN formalization and verification using maude. In: Behaviour Modelling-Foundations and Applications. ACM, pp. 1–12.

Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K., 2009. Instantaneous soundness checking of industrial business process models. In: BPM. In: LNCS, 5701. Springer, pp. 278–293.

Fellman, M., Zasada, A., 2016. State-of-the-art of business process compliance approaches: a survey. In: International Workshop on Enterprise Modeling and Information Systems Architectures. CEUR-WS.org, pp. 60–63.

Gemino, A., Wand, Y., 2005. Complexity and clarity in conceptual modeling: comparison of mandatory and optional properties. Data Know. Eng. 55 (3), 301–326.

Groefsema, H., Bucur, D., 2013. A survey of formal business process verification: From soundness to variability. In: Business Modeling and Soft Design, pp. 198–203.

Huai, W., Liu, X., Sun, H., 2010. Towards trustworthy composite service through business process model verification. In: Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing. IEEE, pp. 422–427.

Jakumeit, E., Buchwald, S., Kroll, M., 2010. Grgen. net. Int. J. Software Tools Technol. Trans. 12 (3–4), 263–271.

Kheldoun, A., Barkaoui, K., Ioualalen, M., 2015. Specification and verification of complex business processes high-level petri net-based approach. In: BPM. In: LNCS, 9253. Springer, pp. 55–71.

Kheldoun, A., Barkaoui, K., Ioualalen, M., 2017. Formal verification of complex business processes based on high-level petri nets. Inf Sci (Ny) 385, 39–54.

Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J., 2013. On structured workflow modelling. In: Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE. In: LNCS, 1789. Springer, pp. 431–445.
Klai, K., Tata, S., Desel, J., 2009. Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes. In: International Conference on Business Process Management. In: LNCS, 5701. Springer, pp. 294–309.
Koehler, J., Tirenni, G., Kumaran, S., 2002. From business process model to consistent implementation: a case for formal verification methods. In: Enterprise Distributed Object Computing Conference. IEEE, pp. 96–106.
Koniewski, R., Dzielinski, A., Amborski, K., et al., 2006. Use of petri nets and business processes management notation in modelling and simulation of multimodal logistics chains. In: European Conference on Modeling and Simulation, pp. 28–31.
Kossak, F., Illibauer, C., Geist, V., Kubovy, J., Natschläger, C., Ziebermayr, T., Kopetzky, T., Freudenthaler, B., Schewe, K.-D., 2014. A Rigorous Semantics for BPMN 2.0 Process Diagrams. Springer International Publishing, pp. 29–152.
Kunze, M., Berger, P., Weske, M., 2012. BPM academic initiative - fostering empirical research. In: BPM (Demo Track), 940, pp. 1–5.
Kunze, M., Luebbe, A., Weidlich, M., Weske, M., 2011. Towards understanding process modeling the case of the BPM academic initiative. In: Business Process Model and Notation. In: LNBIP, Vol. 95. Springer, pp. 44–58.
Kwantes, P.M., Van Gorp, P., Kleijn, J., Rensink, A., 2015. TowardsCompliance Verification Between Global and Local Process Models. In: Graph Transformation, 9151. Springer, pp. 221–236. http://link.springer.com/10.1007/978-3-319-21145-9_14
Laue, R., Mendling, J., 2008. The impact of structuredness on error probability of process models. In: Information Systems and e-Business Technologies. In: LNBIP, Vol. 5. Springer, pp. 585–590.
Lohmann, N., Verbeek, E., Dijkman, R.M., 2009. Petri net transformations for business processes - a survey. Trans. Petri Nets Other Model. Concurren. 2, 46–63.
Martens, A., 2005. Analyzing web service based business processes. In: International Conference on Fundamental Approaches to Software Engineering. In: LNCS, 3442. Springer, pp. 19–33.
Matthias, K., Mathias, W., 2016. Behavioural Models - From Modelling Finite Automata to Analysing Business Processes. Springer doi:10.1007/978-3-319-44960-9.
Mendling, J., 2007. Detection and Prediction of Errors in EPC Business Process Models. Wirtschaftsuniversität Wien Vienna.
Mendling, J., Reijers, H.A., Recker, J., 2010. Activity labeling in process modeling: empirical insights and recommendations. Inf. Syst. 35 (4), 467–482.
Morimoto, S., 2008. A survey of formal verification for business process modeling. In: Computational Science. In: LNCS, 5102. Springer, pp. 514–522.
Muehlen, M., Recker, J., 2008. How much language is enough? Theoretical and practical use of the business process modeling notation. In: CAiSE. In: LNCS, 5074. Springer, pp. 465–479.
Murata, T., 1989. Petri nets: properties, analysis and applications. IEEE Proc. 77 (4), 541–580. doi:10.1109/5.24143.
OMG, 2011. Business Process Model and Notation (BPMN V 2.0).
Polyvyanyy, A., Bussler, C., 2013. The structured phase of concurrency. In: Seminal Contributions to Information Systems Engineering. In: LNCS, 1789. Springer, pp. 257–263.
Polyvyanyy, A., García-Bañuelos, L., Dumas, M., 2012. Structuring acyclic process models. Inf. Syst. 37 (6), 518–538. doi:10.1016/j.is.2011.10.005.
Polyvyanyy, A., Garcia-Banuelos, L., Fahland, D., Weske, M., 2014. Maximal structuring of acyclic process models. Comput. J. 57 (1), 12–35.
Prandi, D., Quaglia, P., Zannone, N., 2008. Formal analysis of BPMN via a translation into COWS. In: Coordination Models and Languages. In: LNCS, 5052. Springer, pp. 249–263.
Puhlmann, F., 2007. Soundness verification of business processes specified in the Pi-Calculus. In: OTM. In: LNCS, 4803. Springer, pp. 6–23.
Puhlmann, F., Weske, M., 2005. Using the π-calculus for formalizing workflow patterns. In: International Conference on Business Process Management. In: LNCS, 3649, pp. 153–168.
Puhlmann, F., Weske, M., 2006. Investigations on soundness regarding lazy activities. In: Business Process Management. In: LNCS, 4102. Springer, pp. 145–160.
Ramadan, M., Elmongui, H.G., Hassan, R., 2011. BPMN formalisation using coloured petri nets. In: International Conference on Software Engineering & Applications.
Rozenberg, G., Engelfriet, J., 1998. Elementary net systems. In: Lectures on Petri Nets I: Basic Models. Springer, pp. 12–121. doi:10.1007/3-540-65306-6_14.
Schmidt, K., 2000. LoLA: a low level analyser. In: ICATPN. In: LNCS, 1825. Springer, pp. 465–474.
Van Der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.T., 2011. Soundness of workflow nets: classification, decidability, and analysis. Formal Aspect. Comput. 23 (3), 333–363.
Van Gorp, P., Dijkman, R., 2013. A visual token-based formalization of BPMN 2.0 based on in-place transformations. Inf. Softw. Technol. 55 (2), 365–394.
Weske, M.,. Business Process Management: Concepts, Languages, Architectures, Second Edition. Springer.
Wong, P.Y.H., Gibbons, J., 2008. A process semantics for BPMN. In: Formal Methods and Software Engineering. In: LNCS, 5256. Springer, pp. 355–374.
Wong, P.Y.H., Gibbons, J., 2011. Formalisations and applications of BPMN. Sci. Comput. Program. 76 (8), 633–650.
Ye, J., Song, W., 2010. Transformation of BPMN diagrams to YAWL nets. J. Softw. 5 (4).