Safeness and Soundness Checking in BPMN 2.0 Collaborations with Sub-Processes

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

School of Science and Technology, University of Camerino, Italy

Abstract. BPMN collaboration models are commonly used to describe internal behaviour and interactions among different participants with a shared objective. An important role in this kind of models is played both by the *message flow*, that permits to describe interactions among participants, and by *sub-process* notation elements, that permits to hide and postpone the definition of fine grained details related internal behaviour. The inherent complexity of modeling participant processes, as well as their interactions, can lead to the violation of fundamentals properties. In this paper, we face the problem of checking the correctness of collaboration models with respect to safeness and soundness properties, when sub-processes and message exchanges are included. To enable verification, we provide an operational semantics for BPMN implemented in a formal framework integrated with the Camunda modelling environment. The resulting tool, named S^3, has been validated using more than two thousand models available on a publicly accessible repository, so to demonstrate its benefits in identifying correctness issues on real models.

1 Introduction

Modelling of complex systems generally requires to provide descriptions of the system at different levels of abstraction. This is important from various perspectives. On the one hand, the modeller can better manage the complexity he/she has to handle going from more 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, abstraction permits to keep models less crowded and more understandable to possible interested readers, thus keeping the information flow in line with general comprehension capabilities [1]. In this regard, BPMN 2.0 offers *collaboration* diagrams that provide a way to represent both a global view on the *message exchange* among different participants, and local details on the behaviour of each single participant by means of *process* specifications. In addition, the notation provides the *sub-process* element to represent a compound activity that can be expanded or collapsed at modeller convenience.

Surprisingly, the support for the sub-process element is not that advanced in most modelling environments used in practice [2]. In particular modelling tools do not provide a valid support for correctness checking when a sub-process is included in a collaboration specification. Indeed the sub-process element cannot just be considered syntactic sugar due to the notion of completeness reported

in the standard specification. In particular, this requires the absence of pending tokens within the sub-process when the end event is reached. This characteristic has a relevant impact on the property of soundness in particular when messages are exchanged by elements embedded in the sub-process. The result is that, according to the standard, flattening a model so to include the description of a sub-process at an higher level, will not provide correct results in relation to verification activities.

In this paper, we face this problem providing a framework enabling the verification of general properties such as *safeness* [3] and *soundness* [4] for collaboration diagrams taking into account specific characteristics introduced by message flow and sub-process elements. In this respect we introduce a novel property, named *message-disregarding soundness*, that relaxes the usual soundness notion by considering sound also those collaborations in which asynchronously sent messages are not handled by the receiver (hence, disregarding possible messages that when the execution completes are still enqueued). We also provide a concrete implementation of the proposed formal framework, called $\mathcal{S}^3$ (*Safeness, Soundness and message-disregarding Soundness verifier*), which we have integrated in the *bpmn.io* modelling environment (*https://bpmn.io/*) to provide an easy access to the verification functionalities we propose here.

To sum up, the major contributions of this paper are:

1. the extension of safeness and soundness properties with the introduction of *message-disregarding soundness*;
2. the definition and the implementation in Java of a formal operational semantics for BPMN collaborations considering sub-processes and message exchange, enabling then the *formal analysis* of BPMN models;
3. a tool for checking safeness and soundness, accessible both as a RESTful service, and as a web-application based on the Camunda modeller.

It is worth to mention that the proposed verification framework, and the corresponding tool, has been extensively validated using models publicly available on a freely accessible process repository (*http://bpmai.org/*) [5].

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

2 Motivation

The verification of BPMN collaboration models has been rather overlooked so far. To the best of our knowledge in literature there are no comprehensive studies on the impact of message exchanges in relation to property verification. This is particularly true taking into account that collaborations are asynchronous, and that properties interaction with sub-processes can lead to subtle effects. In BPMN the expected behaviour of a sub-process element goes through a simple sequence of states such as activation, execution and completion. In case there are messages incoming or outgoing from the sub-process element, these will be managed by the embedded process specification, while it is executed. When activated

a sub-process passes the control flow to the embedded process fragments, and till the process is ongoing the sub-process will stay in the execution state. The termination of the embedded process leads to the completion of the sub-process element, and then the control flow is passed to the next coming element in the model.

At a first glance it seem that verification activities could be performed in a standard way, just after having flattened the internal details of a sub-process with the rest of the model, and without any additional precaution. Unfortunately, this is not the case since the BPMN standard specification equips the sub-process element with its own semantics. In particular, according to the BPMN standard, a sub-process completes only when all the internal tokens are consumed, and then just one token is propagated along the including process. This behaviour would not correspond to a simple flattening that could lead to the masking of unsound behaviour when combined with message exchange, as detailed in the next. Moreover, the consideration of sub-processes and messages in relation to verification activities can generate new scenarios of inconsistency, for which we provide in the following a novel characterization.

We report here a simple example that can help to understand the different situations that could arise in case no verification support is provided. Fig. 1(A) shows a collaboration model in which two pools interacts via a message exchange, while Fig 1(B) shows a flattening of the process in pool A that even if it results quite similar, it implements a completely different behaviour. As it can be observed from Fig. 1(A), the sub-process presents a termination issue, because when a token will reach the end event another token remains in the sub-process. This, however, does not affect the completion of the overall process in *Pool A*, as only one token is produced by the sub-process element on its outgoing edge when all tokens reach the end event in the sub-process. Such termination issue instead remains in the flatted process model in Fig. 1(B). Nevertheless, when considering the collaboration with *Pool B*, there will be a message from *Pool A* to *Pool B* that will never be received.

Now, let us modify the model by reversing the direction of the message, so that it will be directed from *Pool B* to *Pool A*. In such a case the sub-process will even experience a deadlock, since the receiving task will be executed twice and the second time no message will arrive, thus blocking the execution of the enclosing process.

The examples are intentionally un-structured and kept simple, but they contribute to clarify the motivations of our work. In real contexts these situations can easily arise when complex scenarios are considered. Indeed, even if struc-

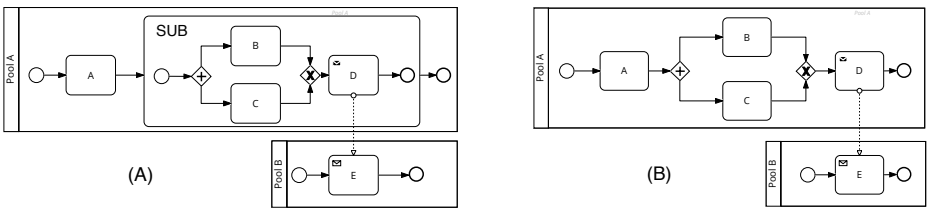

Fig. 1. BPMN unsafe models

turedness has been considered as a good modelling practice [6], designers often do not follow such a guideline [7], because in this way the modelling activity results to be less complex [8] and more expressive [9, 10].

3 Formal Framework

This section presents our formalisation of the BPMN semantics and of correctness properties defined on top of it. Specifically, we first present the syntax and operational semantics we defined for a subset of BPMN elements. This subset includes the elements for describing message exchanges and sub-processes, which play a key role in our work, while are usually not considered or over-abstracted by other formalisations. In selecting the other elements we followed a pragmatic approach, we focused on those regularly used to design process models in practice [11]. However, extending the framework to include further elements is not particularly challenging (even a tricky element as the OR-join can be conveniently added to our formalisation, see [12]). We exploit this formal characterisation to define the notions of safeness and (two variants of) soundness for BPMN collaborations. Notably, as it is the case for similar works, we abstract from data values as we aim at exhaustively analysing all possible executions of a given model. This may certainly lead to counterexamples that will not occur when data are added to the collaboration models. In any case these counterexamples could be useful for modellers that need to refine models adding data characterizations. For the sake of presentation, some definitions, demonstrations, or part of them, are available in the appendix [13].

3.1 Syntax of BPMN Collaborations

To enable the formal treatment of collaborations' semantics, we defined a BNF syntax of their model structure (Fig. 2). 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.

The reader could correctly notice that our syntax is too permissive with respect to the BPMN notation, as it allows to write collaborations that are not admitted by BPMN. However, constraining the syntax, for instance to distinguish processes and collaborations with different syntactic categories, would complicate the definition of the formal semantics. On the other hand, this should not be considered a problem since for modelling activities we rely on an external modelling environment that permits to produce only syntactically correct models, with respect to the standard BPMN syntax.

In the following $\mathsf{e} \in \mathbb{E}$ denotes a *sequence edge*, while $E \in 2^{\mathbb{E}}$ a set of edges; we require $|E| > 1$ when E is used in joining and splitting gateways. Similarly, we require that an event-based gateway should contain at least two message

$$
\begin{aligned}
C ::= {} & \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \mid \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \mid \mathsf{andSplit}(\mathsf{e}_i, E_o) \mid \mathsf{xorSplit}(\mathsf{e}_i, E_o) \\
& \mid \mathsf{andJoin}(E_i, \mathsf{e}_o) \mid \mathsf{xorJoin}(E_i, \mathsf{e}_o) \mid \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok})) \\
& \mid \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o) \mid \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \\
& \mid \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o) \mid C_1 | C_2
\end{aligned}
$$

Fig. 2. Syntax of BPMN collaboration structures

events, i.e. $k > 1$ in each $\mathsf{eventBased}$ term. For reader convenience we refer with e_i to the edge incoming in 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, $\mathsf{m} \in \mathbb{M}$ denotes a *message edge*, enabling message exchanges between pairs of participants in the collaboration. Both e and m denote names uniquely identifying a sequence edge and a message edge, respectively. 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.
- $\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$ represents an end event with an incoming edge e_i and a completing edge e_{cmp}.
- $\mathsf{subProc}(\mathsf{e}_i, C, \mathsf{e}_o)$ represents a sub-process element with incoming edge e_i and outgoing edge e_o. When activated, the enclosed sub-process C behaves according to the elements it consists of, including sub-process elements.
- $C_1 | C_2$ 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.

Since we consider collaborations as a nested structure, to refer the start events of each C we resort to function $start(C)$, which returns their enabling edges:

$$start(C_1 \mid C_2) = start(C_1) \cup start(C_2) \qquad start(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)) = \{\mathsf{e}_{enb}\}$$

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

Notably, we assume that each process in the collaboration has only one start. Thus, the above function applied on the whole collaboration will return as many edges as the number of participants involved in the collaboration, while the application of the function in each 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:

$$end(C_1 \mid C_2) = end(C_1) \cup end(C_2) \qquad end(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})) = \{\mathsf{e}_{cmp}\}$$

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

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. 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; so that $\delta(\mathsf{m}) = \mathsf{n}$ means that there are n messages of type m sent by a participant to another that have not been received yet. Update for δ are defined in a way similar to σ's definitions. Moreover, 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 messages 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 $\forall\ \mathsf{m} \in \mathbb{M}\ .\ \delta(\mathsf{m}) = 0$.*

The operational semantics is defined, as usual, by means of a *labelled transition system* (LTS). In our case, this is a triple $\langle \mathcal{C}, \mathcal{L}, \to\rangle$ 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 $\to \subseteq \mathcal{C} \times \mathcal{L} \times \mathcal{C}$ is a *transition relation*. We will write $\langle C, \sigma, \delta\rangle \xrightarrow{l} \langle C, \sigma', \delta'\rangle$ to indicate that $(\langle C, \sigma, \delta\rangle, l, \langle C, \sigma', \delta'\rangle) \in \to$ 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'. Since collaboration execution only affects the current states, and not the collaboration structure, for the sake of readability we omit the structure from the target configuration of the transition. Thus, a transition $\langle C, \sigma, \delta\rangle \xrightarrow{l} \langle C, \sigma', \delta'\rangle$ is written as $\langle C, \sigma, \delta\rangle \xrightarrow{l} \langle \sigma', \delta'\rangle$.

The transition relation over collaboration configurations formalizes the execution of a collaboration in terms of edge and message marking evolution. It is defined by the rules in Fig. 3. Labels l represent computational steps. In particular, $!\mathsf{m}$ 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).

Before commenting on the rules, we introduce the auxiliary functions they exploit. Specifically, function $inc : \mathbb{S} \times \mathbb{E} \to \mathbb{S}$ (resp. $dec : \mathbb{S} \times \mathbb{E} \to \mathbb{S}$), 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. As usual, the update function for δ are defined in a way similar to σ's definitions. We also use the function $zero : \mathbb{S} \times \mathbb{E} \to \mathbb{S}$ that allows updating a state by setting to

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

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

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

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

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

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

$$\langle \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok})), \sigma, \delta\rangle \xrightarrow{?\mathsf{m}_j} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_{oj}), dec(\delta, \mathsf{m}_j)\rangle \quad \begin{array}{l}\sigma(\mathsf{e}_i) > 0,\\ \delta(\mathsf{m}_j) > 0,\\ 1 \le j \le k\end{array} \quad (C\text{-}EventG)$$

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

$$\langle \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{?\mathsf{m}} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), dec(\delta, \mathsf{m})\rangle \quad \begin{array}{l}\sigma(\mathsf{e}_i) > 0,\\ \delta(\mathsf{m}) > 0\end{array} \quad (C\text{-}TaskRcv)$$

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

$$\langle \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma, \delta\rangle \xrightarrow{?\mathsf{m}} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), dec(\delta, \mathsf{m})\rangle \quad \begin{array}{l}\sigma(\mathsf{e}_i) > 0,\\ \delta(\mathsf{m}) > 0\end{array} \quad (C\text{-}InterRcv)$$

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

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

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

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

Fig. 3. BPMN collaboration semantics

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 $zero(\sigma, \{\mathsf{e}\} \cup E)) = zero(zero(\sigma, \mathsf{e}), E)$.

To check the completion of a sub-process, and more in general of processes and collaborations, we exploit the boolean predicate $completed(\langle C, \sigma, \delta\rangle)$. It is defined according to the prescriptions of the BPMN standard, which states that "a process instance is completed if and only if [...] there is no token remaining within the process instance; no activity of the process is still active. For a process instance to become completed, all tokens in that instance must reach an end node" and "a sub-process instance completes when there are no more tokens in the sub-process and none of its activities is still active" [14, pp. 426, 431].

Then, a collaboration completes when all involved processes complete. The fact that in our formalisation we do not provide a specific construct for identifying processes does not raise any issue related to the collaboration completion check, as tokens cannot pass from one process to another, and edge names are unique in the model. Thus, the collaboration/process/sub-process completion can be formalised as follows.

Definition 2. *Let C be a collaboration, having the form $\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \mid C'$, the predicate $completed(C, \sigma, \delta)$ is defined as $(\sigma(\mathsf{e}_{cmp}) > 0 \wedge \sigma(\mathsf{e}_i) = 0 \wedge isZero(C', \sigma))$, where the predicate $isZero(C', \sigma)$ holds if all edges in the elements of C' have zero token in the state σ, except for the completing edges of end events.*

Notably, the completion of a collaboration does not depend on the exchanged messages, and it is defined considering the arbitrary topology of the model, which hence may have one or more end events with possibly more than one token in the completing edges. Finally, we use the function $marked(\sigma, E)$ to refer to the set of edges in E with at least one token in σ.

We now briefly comment on some of the operational rules in Fig. 3. Rule *C-Start* starts the execution of a collaboration/process/sub-process when it has been activated (i.e., the 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. Rule *C-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 *C-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 that in each outgoing edge. Similarly, rule *C-XorSplit* is applied when a token is available in the incoming edge of a XOR split gateway, the rule decrements the token in the incoming edge and increments the token in one of the outgoing edges, non-deterministically chosen. Rule *C-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 *C-XorJoin* is activated every time there is a token in one of the incoming edges, which is then moved to the outgoing edge. Rule *C-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 tokens in the meantime is decreased (i.e., a message from the corresponding queue is consumed). Rule *C-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 *C-TaskRcv* is activated not only when there is a token in the incoming edge, like the one related to simple tasks, but also when there is a message to be consumed. Similarly, rule *C-TaskSnd*, instead of consuming a token sends a message before moving moving the token to the outgoing edge. Rule *C-InterRcv* (resp. *C-InterSnd*) follows the same behavior of rule *C-TaskRcv* (resp. *C-TaskSnd*). Rules *C-SubProc*$_{Start}$ and *C-SubProc*$_{End}$ deal with a sub-process element. The former rule 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, the sub-process behaves as its body till it completes, according to the completion check performed by the rule *C-SubProc*$_{End}$. When this rule is applied, it removes all tokens from sub-process, and adds a token to the outgoing edge of the sub-process. Actually, due to the definition of sub-process completion (Def. 2), only the completing edges of the end events within the sub-process body need to be set to zero. Finally, *C-Int*$_1$ and *C-Int*$_2$ deal with interleaving in a standard way.

3.3 Safeness and Soundness

We now provide a formal definition for the correctness properties we verify on collaboration models.

Safeness refers to the occurrence of no more than one token at the same time along the same sequence edge of each process in the collaboration. The formalisation of the property is based on the auxiliary function $maxMarking(\cdot)$ that given a configuration determines the maximum number of tokens marking the sequence edges of elements in C according to the state σ. Now, a collaboration is defined to be safe if the maximum marking never exceed one along the collaboration execution. We use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$.

Definition 3 (Safe collaborations). *A collaboration C is* safe *if and only if, given σ and δ such that $isInit(C, \sigma, \delta)$, for all σ' and δ' such that $\langle C, \sigma, \delta\rangle \rightarrow^* \langle \sigma', \delta'\rangle$ we have that $maxMarking(C, \sigma') \leq 1$.*

Soundness is a more elaborated property, which is based on the notions of option to complete, no dead activities and proper completion in relation to Petri Nets. In order to check the same properties on collaborations diagrams we redefined them keeping the general meaning of the property. Intuitively, soundness requires that from any reachable configuration it is possible to reach a (completed) configuration where some end events are marked exactly by a single token and all sequence edges are unmarked. However, this notion does not take into account enqueued messages that will never be consumed. This aspect, as mentioned in Sec. 1 and 2 is quite restrictive in case of cycle, not properly closed split, and not safe model, for this reason we provide two notions of soundness: one that requires message queues to be empty for a proper completion, and another that relaxes such a requirement.

Definition 4 (Sound collaboration). *A collaboration C is* sound *if and only if, given σ and δ such that $isInit(C, \sigma, \delta)$, 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 $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, $\forall\, \mathsf{e}_{cmp} \in marked(\sigma'', end(C))$. $\sigma''(\mathsf{e}_{cmp}) = 1$, $isZero(C, \sigma'')$, and $\forall\, \mathsf{m} \in \mathbb{M}$. $\delta''(\mathsf{m}) = 0$.*

Definition 5 (Message-Disregarding Sound collaboration). *A collaboration C is* message-disregarding sound *if and only if, given σ and δ such that $isInit(C, \sigma, \delta)$, 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 $\langle C, \sigma', \delta'\rangle \rightarrow^* \langle \sigma'', \delta''\rangle$, $\forall\, \mathsf{e}_{cmp} \in marked(\sigma'', end(C))$. $\sigma''(\mathsf{e}_{cmp}) = 1$, and $isZero(C, \sigma'')$.*

4 The $\mathcal{S}^3$ Supporting Tool

In this section, we present $\mathcal{S}^3$, the implementation of our verification framework. Firstly, we show how we check properties on the LTS generated by the implementation of the semantics, and then that the checked properties are formally proved to correspond to their definitions given in Sec. 3 (the proofs of these results are relegated to the online appendix [13]). Then, we illustrate the architecture and the main features of the $\mathcal{S}^3$ tool.

From Theory to Practice. As shown in Sec. 3.2, our BPMN operational semantics is defined in terms of an LTS. 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 (i.e., same σ and δ) is already present. In such a case, we connect the transition edge under construction to the existing state. This characteristic of the generated LTS is then exploited (see Def. 10) 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). Considering a collaboration C, we now formally introduce the notions of safeness, soundness and message-disregarding soundness related to the LTS induced by the semantics. Consequently, we prove the correspondence between these definitions and Def. 3, 4 and 5, respectively.

Definition 6 (Safe Labelled Transition Systems). *A 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 $\forall\, \mathsf{e} \in \mathbb{E} \,.\, \sigma(\mathsf{e}) \leq 1$.*

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 one time, and the set of edge labels incoming to the end events of a collaboration.

Definition 7 (Incoming Labels). *Let $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ be an LTS and $\langle C, \sigma, \delta \rangle \in \mathcal{C}$, $incoming(C, \sigma, \delta) = \{l \in \mathcal{L} \mid \exists\, \sigma', \delta' : \langle C, \sigma', \delta' \rangle \xrightarrow{l} \langle \sigma, \delta \rangle\}$.*

Definition 8 (Labels Duplication). *Let $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ be an LTS and $I \subseteq \mathcal{L}$ a set of labels, predicate $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, I)$ holds true if $\forall\, l_i \in I$ 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 $\langle C, \sigma_1, \delta_1 \rangle \xRightarrow{l_i} \langle C, \sigma_2, \delta_2 \rangle \xRightarrow{l_i} \langle C, \sigma_3, \delta_3 \rangle$, where $\xRightarrow{l_i} = \rightarrow^* \xrightarrow{l_i} \rightarrow^*$, never holds.*

Definition 9 (End Events Incoming Labels). *Let C be a collaboration, then $endIn(\cdot)$ is inductively defined as follows:*

$$endIn(C_1 \mid C_2) = endIn(C_1) \cup endIn(C_2) \qquad endIn(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})) = \{\mathsf{e}_i\}$$
$$endIn(C) = \emptyset \;\; \textit{for any element } C \neq \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$$

Now, our notions of soundness on LTSs and their correspondence with the definitions on collaborations can be defined as follows.

Definition 10 (Sound LTSs). *A 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* $\langle C, \sigma, \delta \rangle \xrightarrow{l} \langle \sigma', \delta' \rangle$*)*
(ii) $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, incoming(\langle C, \sigma, \delta \rangle))$
(iii) $incoming(\langle C, \sigma, \delta \rangle) = endIn(C)$
(iv) $\forall\ \mathsf{e} \in \mathbb{E}\ .\ \sigma(\mathsf{e}) = 0$
(v) $\forall\ \mathsf{m} \in \mathbb{M}\ .\ \delta(\mathsf{m}) = 0$

Definition 11 (Message-Disregarding Sound LTSs). *A LTS* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *of a collaboration* C *is* message-disregarding 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) $isNotDuplicated(\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle, incoming(\langle C, \sigma, \delta \rangle))$
(ii) $incoming(\langle C, \sigma, \delta \rangle) = endIn(C)$
(iii) $\forall\ \mathsf{e} \in \mathbb{E}\ .\ \sigma(\mathsf{e}) = 0$

Theorem 1 (Properties correspondence). *Let* C *be a collaboration and* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *its LTS, then:*
(i) C *is safe iff* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *is safe*
(ii) C *is sound iff* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *is sound*
(iii) C *is message-disregarding sound iff* $\langle \mathcal{C}, \mathcal{L}, \rightarrow \rangle$ *is message-disregarding sound*

Implementation.

$\mathcal{S}^3$ is based on a client-server architecture composed by a client, developed in HTML/Javascript and embedding the Camunda *bpmn.io* modeller, and a RESTful-service developed in Java. The service embeds the core of the system, and it takes in input the *.bpmn* representation of a BPMN model, and then it sends back the results of properties checking. The received model is parsed by $\mathcal{S}^3$ using the Camunda APIs so to derive the corresponding LTS, and according to the semantics defined in Sec. 3.2. In case of a property violation the service returns the corresponding counterexamples.

Fig. 4 depicts the $\mathcal{S}^3$ modelling environment, which we make available as a web application (http://pros.unicam.it:8080/S3/modeler/). Using the graphical interface the modeller can design 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 will be shown using positive/negative icons next to the property names. In case of a negative

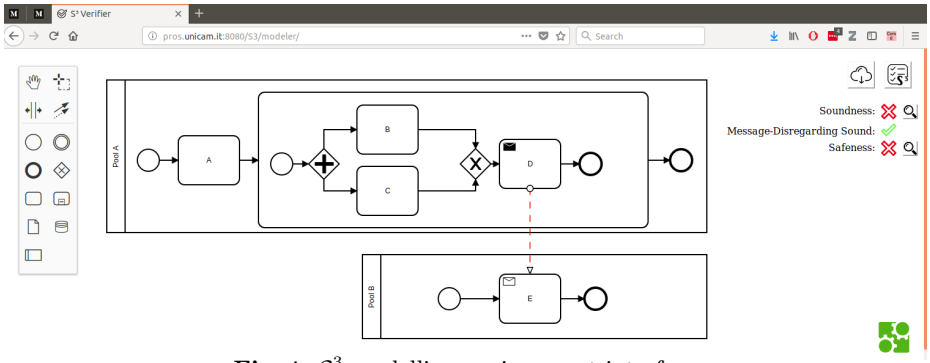

Fig. 4. $\mathcal{S}^3$ modelling environment interface

outcome, an additional button 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. It is also possible to export the model as a *.bpmn* file. Finally the service can also be invoked by other clients, so to make easier its integration with other modelling tools.

5 Validation

This section presents the experimentation we ran using $\mathcal{S}^3$. The validation has been shaped considering the following research questions:

- Are safeness, soundness and message-disregarding soundness correctly handled by modellers, or do they release models violating such properties?
- Can $\mathcal{S}^3$ effectively support modellers in the verification of their models?

Experimentation Set-Up. In order to validate our verification framework against real-word processes, we consider BPMN collaboration models provided by the BPM Academic Initiative (http://bpmai.org/) [5]. This is a collection of models codified using various process modelling languages, and particularly suited to investigate modelling practices [15]. The raw dataset consists of 16 032 BPMN models, but we restricted to the latest revision of the models with at least 5 elements and 100% of connectedness[1]. A model without this level of connectedness includes disconnected fragments, which typically means that the model has not been finalized, yet. On the other hand 5 is the minimum number of elements to have a simple collaboration. This gave us a dataset of 2 441 models. Considering our reference dataset, we perform a preliminary transformation step from .json (the repository format) to .bpmn (the format we manage), and then we check safeness, soundness and message disregarding soundness.

Experimentation Results. Table 1 reports the results of our validation. In particular, we cluster the models according to the number of BPMN elements they contain, and given that only three models in our data set have more than fifty elements the last class we consider is that 40–49. 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 more than 98%), 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 disregarding soundness properties the situation is rather worse. Not surprisingly we notice that the percentage of correct models decreases as the model size increases. On the total only 58% of the model are sound and only 61% are message-disregarding sound. The more relaxed form of soundness does not distinguish a high number of models, nevertheless the percentage (3% around 60 models) can justify its use in practice. At the same time it is evident that soundness is not an easy property to satisfy in particular when the model reach a certain dimension. Therefore the availability of tools for automatic verification

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

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. We report only one number since the three properties are verified in a single visit of the resulting LTS model. As it can be observed, the verification time slightly increases with the dimension of the model. Nevertheless the observed data show that the considered properties can be verified in reasonable time on a real dataset. The data on the class 40-49 did not correspond to our expectation so we manually checked the model and we discovered that the 9 models indeed are not much interesting and some of them seem to be kind of replicas. For such a reason the last line in the table is not really meaningful.

Class	*Mod.*	*Safe*	*Sound*	*MD Sound*	*T.Min*	*T.Max*	*T.Avg*	*Std.Dev.*
5–9	1372	1351 (98,46%)	869 (63,33%)	904 (65,88%)	0,009	6,761	0,020	0,186
10–19	910	883 (97,03%)	462 (50,76%)	487 (53,51%)	0,011	756,854	1,853	27,953
20–29	137	134 (97,81%)	51 (37,22%)	57 (41,60%)	0,015	1110,885	15,569	100,927
30–39	13	13 (100%)	4 (30,77%)	4 (30,77%)	0,028	1945,761	149,735	539,638
40–49	9	9 (100%)	3 (33,33%)	3 (33,33%)	0,025	0,490	0,112	0,147

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

A high variability on the time needed can be observed in relation to the classes "20–29" and "30–39". 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. In particular, verification activities are particularly affected by the presence of notation elements leading to the interleaving, such as parallel or pool statements, that are not always included in models. 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 repository.

Summing up, answering to the research questions posed at the beginning of this section, 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. In addition, the experiments show that the proposed approach seems to be applicable in practice to realistic BPMN models.

6 Related Work

Much effort has been devoted to the formalisation and verification of business processes [16–18], but to the best of our knowledge there is not a research work satisfying all the following requirements: (i) provide a direct formalisation taking into account message exchange and sub-process behaviour, as well their impact on collaborations; (ii) provide a relaxed variant of soundness to distinguish issues related to the process internal behaviour from those related to message exchange; (iii) provide a tool support integrating modelling activity with the verification of safeness and soundness properties.

Soundness and safeness are certainly considered the most important general correctness criteria for business processes [19], and different definitions can be found in the literature in relation to different process languages, such as for instance Workflow Nets [20], EPC [21], and Global Interaction Nets [22]. Also relaxed notions of soundness, allowing for instance for the occurrence of deadlocks and livelocks, are reported in literature. For instance in [23] the authors introduce the definition of lazy soundness for Workflow Nets. However, none of the relaxed notions we found in the literature is related to communication aspects.

For BPMN models the definition of soundness and safeness is generally reconducted to that of the formalism used to provide the encoding semantics for the notation. In this respect notable is the work in [24], where the authors provide definitions for the mentioned properties just in natural language, and correspondingly they provide a mapping in Petri Nets for BPMN 1.1.

Also Process calculi has been considered to formalise BPMN. In particular, in [25] a translation into a CSP-like language for a subset of well-formed BPMN process diagrams is proposed. However, the objective is consistency checking of business process models at different levels of abstraction (i.e. refinement checking). In particular, no definition for safeness and soundness is reported.

An encoding from BPMN to Recursive ECATNets is defined in [26]. These nets are expressed in terms of rewriting logic and given in input to the Maude LTL model checker. The approach suffers from the already discussed issues related to encodings and in particular does not distinguish the message flow on the encoded model. Differently from our semantics, none of the available encodings of BPMN in Petri Nets, or similar formalisms, permit to distinguish sequence and message edges, and richer notations should be considered. Correspondingly analysis activities based on such encodings are not able to differentiate such issues.

In our case the definition of a direct formalization for the BPMN notation permits to provide definitions for relevant properties, such as soundness and safeness, on the base of 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 unsafe, and then verification activities should be able to distinguish such situations.

Our contribution is certainly related to [27] where a formalisation of BPMN processes based on in-place graph transformation rules is provided. The approach is meant to enable compliance verification between global and local process models, thanks to the introduction of a transformation of the collaboration diagrams message flow into LTL formulas [28]. Differently in our work we provide an operational semantics in terms of LTS, so to reuse verification techniques well-established for this kind model, and we aim at verifying general properties of models. Another attempt to propose a direct formalisation of soundness on BPMN has been described in [29]. However, the paper discusses soundness only for well-formed BPMN models without providing any concrete tool. Finally, in [30] some of us proposed a direct formalisation and a verification approach for

BPMN collaboration. The approach was implemented in the BProVe tool [31] permitting to concretely check soundness of BPMN models. However these works did not consider the impact of the message flow, and of sub-process elements on safeness and soundness.

7 Concluding Remarks and Future Works

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 on a subset of elements of the BPMN collaborations, including in particular sub-processes and messages. The compositional nature of the approach permits its easy extension to include additional elements. On top of the defined semantics we established the notion of safeness and soundness for BPMN models, also including the possibility to consider or not the impact of message exchange on soundness. The semantics and the property verification have been implemented in $\mathcal{S}^3$ tool available as an on-line service, and that has been integrated with the *bpmn.io modeller*, so to provide a fully integrated modelling and verification environment. The usage of the tool and its real effectiveness have been checked using an open repository, that permitted to run a validation using more than 2.4K models.

In the future, we plan to extend our framework to other BPMN relevant characteristics, such as data, exceptions, and time management, so to extend the analysis for BPMN models. Correspondingly we intend to proceed with further validation campaigns of the implemented tool.

References

1. Corradini, F., Ferrari, A., Fornari, F., Gnesi, S., Polini, A., Re, B., Spagnolo, G.: A guidelines framework for understandable BPMN models. Data Knowl. Eng. **113** (2018) 129–154
2. Corradini, F., Polini, A., Re, B., Rossi, L., Tiezzi, F.: Supporting multi-layer modeling in BPMN collaborations. In: EOMAS@CAiSE. Volume 298 of LNBIP. Springer (2017) 53–67
3. van der Aalst, W.: Workflow verification: finding control-flow errors using Petri-Net-Based Techniques. In: BPM. Volume 1806 of LNCS. Springer (2000) 161–183
4. Prinz, T.M.: Fast soundness verification of workflow graphs. In: ZEUS. Volume 1029 of LNCS., Springer (2013) 31–38
5. Kunze, M., Berger, P., Weske, M.: BPM Academic Initiative - Fostering Empirical Research. In: BPM (Demo Track). Volume 940. (2012) 1 – 5
6. Laue, R., Mendling, J.: The impact of structuredness on error probability of process models. In: Information Systems and e-Business Tech. Volume 5. (2008) 585–590
7. Polyvyanyy, A., Bussler, C.: The structured phase of concurrency. In: Seminal Contributions to Information Systems Engineering. Springer (2013) 257–263
8. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J.: On structured workflow modelling. In: Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE. Volume 9539 of LNCS. Springer (2000) 431–445
9. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Information Systems **37**(6) (September 2012) 518–538

10. Polyvyanyy, A., Garcia-Banuelos, L., Fahland, D., Weske, M.: Maximal Structuring of Acyclic Process Models. The Computer Journal **57**(1) (January 2014) 12–35
11. Muehlen, M., Recker, J.: How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: CAiSE. Volume 5074 of LNCS. Springer (2008) 465–479
12. Corradini, F., Muzi, C., Re, B., Rossi, L., Tiezzi, F.: Global vs. Local Semantics of BPMN 2.0 OR-Join. In: SOFSEM. Volume 10706 of LNCN. (2018) 321–336
13. Corradini, F., Morichetta, A., Polini, A., Re, B., Rossi, L., Tiezzi, F.: Safeness and soundness checking in BPMN 2.0 collaborations - APPENDIX. http://pros.unicam.it/s3/ (2017)
14. OMG: Business Process Model and Notation (BPMN V 2.0) (2011)
15. Kunze, M., Luebbe, A., Weidlich, M., Weske, M.: Towards understanding process modeling the case of the BPM academic initiative. In: Business Process Model and Notation. Volume 95 of LNBIP. Springer (2011) 44–58
16. Morimoto, S.: A Survey of Formal Verification for Business Process Modeling. In: Computational Science. Volume 5102. Springer (2008) 514–522
17. Groefsema, H., Bucur, D.: A survey of formal business process verification: From soundness to variability. In: Business Modeling and Soft Design. (2013) 198–203
18. Fellman, M., Zasada, A.: State-of-the-Art of Business Process Compliance Approaches: A Survey. In: Information Systems. (2014)
19. Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. In: Found of Sw Tech & Theor Comp Science. Volume 761 of LNCS. Springer (1993) 326–337
20. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects of Computing **23**(3) (2011) 333–363
21. Mendling, J.: Detection and prediction of errors in EPC business process models. PhD thesis, Wirtschaftsuniversität Wien Vienna (2007)
22. Roa, J., Chiotti, O., Villarreal, P.: A verification method for collaborative business processes. In: BPM. Volume 99 of LNBIP. Springer (2011) 293–305
23. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In: Business Process Management. Volume 4102. Springer (2006) 145–160
24. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology **50**(12) (2008) 1281–1294
25. Wong, P.Y., Gibbons, J.: Formalisations and applications of BPMN. Science of Computer Programming **76**(8) (2011) 633–650
26. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal verification of complex business processes based on high-level Petri nets. Information Sciences **385/6** (2017) 39–54
27. Van Gorp, P., Dijkman, R.: A visual token-based formalization of BPMN 2.0 based on in-place transformations. Information and Soft. Tech. **55**(2) (2013) 365–394
28. Kwantes, P.M., Van Gorp, P., Kleijn, J., Rensink, A.: Towards Compliance Verification Between Global and Local Process Models. In: Graph Transformation. Volume 9151. Springer (2015) 221–236
29. El-Saber, N., Boronat, A.: BPMN Formalization and Verification Using Maude. In: Behaviour Modelling-Foundations and Applications, ACM (2014) 1–12
30. Corradini, F., Polini, A., Re, B., Tiezzi, F.: An operational semantics of BPMN collaboration. In: FACS. Number 9539 in LNCS, Springer (2015) 161–180
31. Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., Vandin, A.: BProVe: a Formal Verification Framework for Business Process Models. In: 32th International Conference on Automated Software Engineering, IEEE (2017) 217–228