A Classification of BPMN Collaborations based on Safeness and Soundness Notions

Flavio Corradini, Chiara Muzi, Barbara Re, and Francesco Tiezzi

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

Abstract. BPMN has a huge uptake in modelling business process collaborations in both academia and industry. It results that providing a solid foundation to enable BPMN designers to understand their models in a consistent way is becoming more and more important. In our investigation we define and exploit a formal characterisation of the collaborations' semantics, specifically and directly given for BPMN models, to provide a classification of BPMN collaborations. In particular, we refer to collaborations involving processes with arbitrary topology, thus overcoming the well-structuredness limitations. The proposed classification is based on some of the most important correctness properties, namely safeness and soundness. We prove, with a uniform formal framework, some conjectured and expected results and, most of all, we achieve novel results for BPMN collaborations concerning the relationships between safeness and soundness, and their compositionality, that represent major advances in the state-of-the-art.

1 Introduction

For some years now, organisations recognised the importance of having tools to model how to behave in order to reach the organisation objectives. This is generally reflected in a business process model that is characterised as "*a collection of related and structured activities undertaken by one or more organisations in order to pursue some particular goal. [. . .] BPs are often interrelated since the execution of a business process often results in the activation of related business processes within the same or other organisations*" [1]. Up to now, several languages have been proposed to represent business process models. However, BPMN [2] is the standard language since it is widely accepted in both academia and industry. In particular, the BPMN collaboration model is used to describe distributed and complex scenarios where multiple participants interact via message exchange.

Even if widely accepted, BPMN major drawback is related to the possible misunderstanding of its execution semantics defined by means of natural text descriptions, sometimes containing misleading information [3]. To fill this gap, much effort has been devoted to formalise BPMN semantics by means of mapping to other formal languages. The most relevant is the one to Petri Nets provided by Dijkman et al. [4]. However, models resulting from mapping inherits constraints given by the target formal language and none of them considers BPMN specificities such as the different abstraction levels (i.e., collaboration, process, and sub-process), the asynchronous communication model, and the notion of completeness adapted to different types of event (i.e., simple, message throwing, and terminate).

Our investigation is based on a formal characterisation of the BPMN semantics specifically given for collaboration models. It is used to formally define a classification of these models according to relevant properties of the business process domain. It is worth noticing that we do not aim at providing a generic classification approach suitable to different kinds of workflow models, but we specifically focus on the BPMN notation. To this aim, our formal semantics is directly defined on BPMN elements rather than as an mapping to other formalisms. Our intention is to introduce solid foundation to enable BPMN designers to understand their models, and properties they enjoy, in a consistent way. Relying on this classification, systematic methodological advice for modelling BPMN diagrams can be provided, thus avoiding errors during their enactment.

As a distinctive aspect, the proposed semantics supports models with arbitrary topology, including of course also well-structured ones. This is necessary to enable the classification of both structured and unstructured models with respect to the considered properties. Moreover, studying unstructured models is motivated by the fact that model structuredness can only be achieved at the expense of increased model size [5], or it cannot be applicable at all [6] [7], and most of all it would anyway limit BPMN designer freedom [8]. The use of unstructured models in practice, especially when the model size increases, is also confirmed by a study we conducted [9] considering the public repository of BPMN models provided by the BPM Academic Initiative (`http://bpmai.org`).

In terms of properties, our classification is based on *safeness* [10] and *soundness* [11] [12]. Despite the large body of work on BPMN, no formal definition of such properties directly given on BPMN is provided. The common practice is to reconsider in the business process domain some studies related to formal properties on Petri Nets [13], Workflow Nets [10], and Elementary Nets [14]. We believe instead that the direct formal characterisation we provide for such properties considering a unique framework is crucial, as it does not leave any room for ambiguity, and increases the potential for formal reasoning on BPMN. In this respect, a further contribution is the introduction of a variant of the soundness property, named *message-aware soundness*, suitable to underline that asynchronously sent messages are properly handled by the receiver (hence, avoiding possible pending messages when the execution completes).

The provided classification relies on the considered properties and their relationship. We demonstrate that a well-structured collaboration is always safe, but the vice versa does not always hold. We also prove that well-structuredness implies soundness only at process level, while there are well-structured collaborations that are not sound. Regarding the relationship between soundness and safeness we prove the existence of unsafe models that are sound. Moreover, we study safeness and soundness compositionality. Finally, we show how the use of some BPMN constructs, namely terminate event and sub-processes, crucially affects the model classification, thus moving BPMN models from one class to another.

The rest of the paper is organised as follows. Sec. 2 provides background notions on BPMN and the considered properties. Sec. 3 introduces a first insight into the obtained results, while Sec. 4 introduces the proposed formal framework. Sec. 5 provides the definition of properties, and Sec. 6 makes clearer the relationships between these prop-

erties. Sec. 7 presents the study on safeness and soundness compositionality. Finally, Sec. 8 discusses related works, and Sec. 10 concludes the paper.

2 Background and Basic Notions

In this section we introduce BPMN 2.0 collaboration diagrams presenting a scenario used throughout the paper as a running example. We also illustrate basic properties we consider in the rest of paper.

2.1 A Travel Agency Scenario in BPMN

The considered scenario combines the activities of a Customer and a Travel Agency (Fig. 1). It is used throughout the paper as a running example. This example is intentionally kept simple, as it just aims at illustrating how the language works in practice.

Running Example (1/9). The Travel Agency continuously offers travels to the Customer, until an offer is accepted. If the Customer is interested in one offer, she decides to book the travel and refuses all the other offers that the Travel Agency insistently proposes. As soon as the booking is received by the Travel Agency, it sends back a confirmation message, and asks for the payment of the travel. When this is completed the ticket is sent to the Customer, and the Travel Agency activities immediately end.

The processes of the Customer and of the Travel agency are represented inside two rectangles, called *pools*. These are used to represent participants or organisations involved in a collaboration, and provide details on internal process specifications and related elements. The flow of process elements in the same or different pools is given by *connecting edges*. In particular, sequence edges are used to specify the internal flow of the process, thus ordering elements in the same pool, while message edges are dashed connectors used to visualise communication flows between organisations.

Considering the Customer pool, from left to right, we have that as soon as the process starts, due to the presence of a *start event* (drawn as a circle with an outgoing sequence edge), the Customer checks the travel offer. This is done by executing a *receiving task* (drawn as a rectangle with rounded corners, an incoming message edge, and an incoming and an outgoing sequence edge). Then, she decides either to book

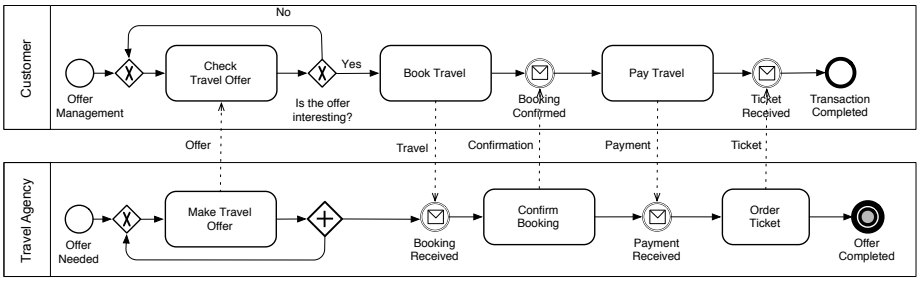


Fig. 1: BPMN collaboration diagram of a Travel Agency scenario.

the travel or to wait for other offers, by means of a cycle composed of two *gateways*: an XOR-Join (drawn as a diamond marked by $\times$ with two incoming sequence edges and an outgoing one) that acts as a pass-through, meaning that it is activated each time the gateway is reached; and an XOR-Split (having two outgoing sequence edges and an incoming one), used after a decision to fork the flow into two branches. After the Customer finds the interesting offer, she books the travel, by sending a message to the Travel Agency by executing a *sending task* (having an outgoing message edge), and waits for the booking confirmation. As soon as she receives the booking confirmation, through an *intermediate receiving event* (drawn as a circle with an incoming message edge), she pays the travel, receives the ticket from the Agency and her specific works terminate by means of an *end event* (drawn as a thick circle).

Considering the work of the Travel Agency, as soon as its process starts, it makes travel offers to the Customer, by means of a *sending task*, until an offer is accepted. Thanks to the behaviour of the AND-Split (drawn as a diamond marked by $+$ with two outgoing sequence edges and an incoming one, used to enable parallel execution flows) combined with the XOR-Join in a cycle, it continuously make offers. At the same time, it proceeds in order to receive a booking via an *intermediate receiving event*. Then, it confirms the booking and sends a notification to the Customer. Finally, after receiving the payment, it orders and sends the ticket, thus completing its activities by means of a *terminate end event* (displayed by a thick circle with a darkened circle inside) which stops and aborts the running process.

There are other BPMN elements we consider in our work that, however, are not present in the running example. In particular, regarding gateways, the AND-Join permits to synchronise the execution of two or more parallel branches, as it waits for all incoming sequence edges to complete before triggering the outgoing flow. Another type of gateway we consider is the *event-based* one: it is similar to the XOR-Split, but its outgoing branches activation depends on taking place of receiving events. For the sake of presentation we do not include the OR gateway, whose semantics would only add complexity to the readability of the work and for our aims it can be treated as an AND gateway, since it affects the studied properties in a similar way. Regarding events, we can have also a *start message event*, which permits to start upon the reception of a message; and an *end message event*, which sends a message and ends the process. Moreover, although the intermediate event in the example shows only the receiving behaviour, this event can also send a message. Our choice of the considered BPMN fragment is driven by practical aspects. Indeed, as shown in [15], even if the BPMN specification is quite wide, only less than 20% of its vocabulary is used regularly in designing business process models. We therefore selected those constructs that are most used in practice.

2.2 Well-structuredness, Safeness and Soundness for BPMN

Being inspired by the Petri Net community we reconsidered *well-structuredness*, *safeness* and *soundness* for BPMN models focusing both to the internal characteristics of a single process in a collaboration, and to the whole collaboration itself. In the following we introduce such properties, while their formalisation is provided in Section 5.

Well-structuredness relates to the way the various model elements are connected with each other. BPMN, as well as most business process modelling notations, allows

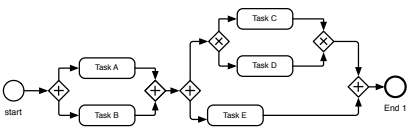


Fig. 2: A WS Process Model.

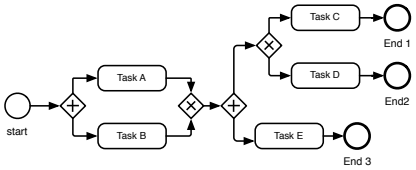


Fig. 3: A non WS Process Model.

models to have arbitrary topology. However, to avoid undesired behaviours, modelling guidelines suggest to use structured building blocks [16], thus obtaining *well-structured* process model [17]. In particular, in a well-structured process (see Def. 4), for every split gateway there is a corresponding join gateway such that the fragment of the model between the split and the join forms a single-entry-single-exit process component. As an example, the process in Fig. 2 is the well-structured version of the unstructured process in Fig. 3. The notion of well-structured can be extended to process collaborations (see Def. 5), requiring that the processes of all involved organisations are well-structured.

Safeness and *Soundness* relate to the way a process model can be instantiated, executed and completed.

A BPMN proces model is *safe* if during its execution no more than one token occur along the same sequence edge (see Def. 7). It is inspired by the Petri-Net scenario where safeness means that the Petri Net does not contain more than one tokens in all reachable markings [18]. Safeness properties naturally extend to process collaborations, requiring that the processes of all involved organisations satisfy it considering the overall collaboration execution (see Def. 8).

A BPMN process model is *sound* if it can complete without lefting over active elements and all the model elements can be activated at least in one of the execution traces (see Def. 10). It is inspired by literature that since the mid nineties presents several version of soundness considering different modelling language [18] [19] [20] [21]. To escape from the jungle of soundness definitions and aiming to capture the BPMN expressibility here we based to the classical soundness definition for processes [22] that informally requires the satisfaction of three sub-properties: *(i) Option to complete*: any running process instance must eventually complete, *(ii) Proper completion*: at the moment of completion, each token of the process instance must be in a different end event, *(iii) No dead activities*: any activity can be executed in at least one process instance. Moreover, considering the BPMN notion of completeness [23, pp. 426, 431] requiring that all tokens in that instance must reach an end node (see Def. 3), we have that soundness is reduced to the satisfaction of one property, i.e. *option to complete*. In fact, the *proper completion* property is included in the definition of successful completion, since it requires that "there exists no related activity of this process which is still running or enabled". Moreover, the *no dead activities* property is equivalent to requiring the complete execution of a process. In fact, the only way to have dead activities is that the incoming sequence flow of that activity is never reached by a token. This can happen either when there is a deadlock upstream the considered activity or when there are some conditions on gateways. The first case is subsumed in the notion of completeness, while the second case is not caught by our semantics since reasoning on models with data is

left to future work. Soundness can be extended to process collaborations (see Def. 11), considering the notion of completeness under the asynchronous nature of the BPMN communication model. Indeed it is not always true that all the incoming messages are available when an activity is to be executed.

3 Classification Results

In this section, we show how BPMN collaborations can be classified according to well-structuredness, safeness, and soundness. Differently from other classifications [24] [25] reasoning at process level thanks to the Workflow Nets semantics, our study directly addresses BPMN collaboration models and this has led to novel results. As we formally prove in the following sections, the obtained results are synthesized in the Euler diagram in Fig. 4, showing that:

(i) all well-structured collaborations are safe, but the vice versa does not hold;
(ii) there are well-structured collaborations that are neither sound nor message-aware sound;
(iii) there are sound and message-aware sound collaborations that are not safe.

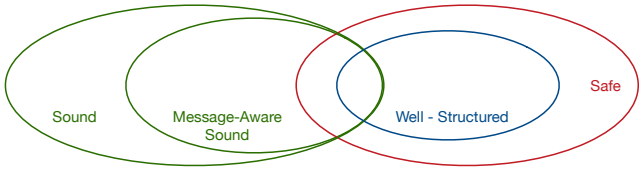


Fig. 4: Classification of BPMN collaborations.

In the following we first discuss how our classification advances the state of the art.

Advances with respect to already available classifications. Result *(i)* demonstrates that well-structured collaborations represent a subclass of safe collaborations. We also show that such relation is valid at process level, where the classification relaxes the existing results on Workflow Nets, stating that a process model to be safe has to be not only well-structured, but also sound [25].

Result *(ii)* shows that there are well-structured collaborations that are not sound. This is also valid at process level confirming results provided on Workflow Nets, where well-structuredness implies soundness [26], and relaxing the one obtained in Petri Nets [24], where relaxed soundness and well-structuredness together imply soundness.

Results *(i)* and *(ii)* together confirm limits of well-structuredness as a correctness criterion. Indeed, only consider well-structuredness is too strict, as some safe and sound models that are not well-structured result discarded right from the start.

Also result *(iii)* shows that there are sound and message-aware sound collaborations that are not safe. This also can be observed at the process level resulting as a novel contribution strictly related to the expressiveness of BPMN and its differences with respect to other workflow languages. In fact, Van der Aalst shows that soundness of a Workflow Net is equivalent to liveness and boundedness of the corresponding short-circuited Petri Net [27]. Similarly, in workflow graphs and, equivalently, free-choice

Petri Nets, soundness can be characterized in terms of two types of local errors, viz. deadlock and lack of synchronization: a workflow graph is sound if it contains neither a deadlock nor a lack of synchronization [28] [29]. Thus, a sound workflow is always safe. In BPMN instead there are unsafe processes that are sound.

Summing up, being collaboration an open field of study, result *(i)* together with result *(ii)* and *(iii)* are novel and influenced also the reasoning at process level. This is mainly due to the effects of the terminate end event and sub-processes behaviour impacting on models classification, both at process and collaborations levels, as shown in the following.

Advance in Classifying BPMN Models. Our BPMN formalisation considers as first-class citizens BPMN specificities including in the semantics collaboration, process and sub-process levels, an asynchronous communication models, and the completeness notion distinguishing the effect of end event and the terminate event.

Considering collaboration models, we can observe pools that exchange message tokens, while in each pool the execution is rendered by the movements of the sequence flow tokens at process level. In this setting, there is a clear difference between the notion of safeness directly defined on BPMN collaborations with respect to that defined on Petri Nets and applied to the Petri Nets resulting from the translation of BPMN collaborations. According to well know mapping, such as the one in [30] and in [31], safeness of a BPMN collaboration only refers to tokens on the sequence edges of the involved processes, while in its Petri Nets translation refers to token both on message and sequence edges. Indeed, such distinction is not considered in the considered mappings, because a message is rendered as a (standard) 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 enqueued 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 are able to consider different notions of soundness according to the requirements we impose on message queues (e.g., ignoring or not pending messages). Again, due to lack of distinction between message and sequence edges, these fine-grained reasoning are precluded using the current translations from BPMN to Petri Nets.

The study of BPMN models via the Petri Nets based frameworks has another limitation concerning the management of the terminate event. Most of the available mappings, such as the ones in [31] and [32], do not consider the termination end event, while in the one provided in [30], terminate events are treated as a special type of error events, which however occur mainly on sub-processes, whose translation assumes safeness. This does not allow reasoning on most models including the terminate event, and more in general on all models including unsafe sub-processes. Anyway, given the local nature of Petri Nets transitions, such cancellation patterns are difficult to handle. This is confirmed in [33], stating that modelling a vacuum cleaner, i.e., a construct to remove all the tokens from a given fragment of a net, is possible but results in a spaghetti-like diagram.

The ability of our formal framework to properly classify BPMN models both at process and collaboration level, together with our treatment of the terminate event and sub-processes without any of the restrictions mentioned above, has led us to provide a more

(a) (b)

Fig. 5: Reasoning at process (a) and collaboration (b) level.

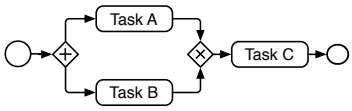


Fig. 6: Unsound process.

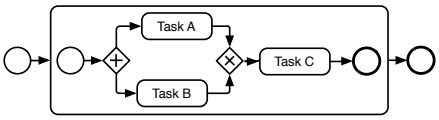


Fig. 7: Sound process with an unsound sub-process.

precise classification of the model as synthesised by the Euler diagrams in Fig. 5(a) and Fig. 5(b).

In particular, Fig. 5(a) underlines reasoning that can be done at process level on soundness. Here it emerges how the terminate event can affect model soundness, as using it in place of an end event may render sound a model that was unsound. For example, let us consider the process in Fig. 6; it is a simple process that first runs in parallel Task A and Task B, then resulting in two possible executions of Task C, and finally completes with an end event. According to the proposed classification the model is unsound. In fact, from any reachable configuration of the process it is not possible to arrive in a (completed) configuration where all marked end events are marked exactly by a single token and all sequence edges are unmarked. Now, let us consider another model, obtained from the one in Fig. 6 by replacing the end event with a terminate event. The resulting model is sound. This is due to the behaviour of the terminate event that, when reached, removes all tokens in the process. We underline that, although the two models are quite similar, in terms of our classification they result to be significantly different. Also the use of sub-processes can impact on the satisfaction of the soundness property. Fig. 7 shows a simple process model where the unsound process in Fig. 6 is included in the sub-process. Notably, a sub-process is not syntactic sugar that can be removed via a sort of macro expansion. Indeed, 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 [23]. Thus, it results that the model in Fig. 7 is sound. Its behaviour would not correspond to that of the process obtained by flattening it, as the resulting model could be unsound. Notice, this reasoning is not affected by safeness and in particular, it cannot be extended to collaborations. In fact, as we discuss in Sec. 7, when we compose two sound processes the resulting collaboration could be either sound or unsound.

Interesting situations also arise when focussing on the collaboration level, as highlighted in Fig. 5(b). Worth to notice is the possibility to transform with a small change an unsound collaboration into a sound one.

In Fig. 8, Fig. 9 and 10 we report a simple example showing the impact of sub-processes. Also in this case the models are rather similar, but according to our classification the result is completely different. The collaboration model in Fig. 8 is both unsound

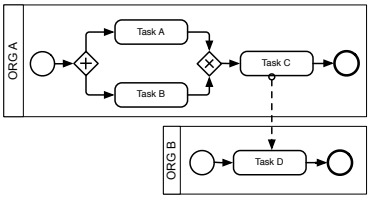


Fig. 8: An example of unsound and message-aware unsound.

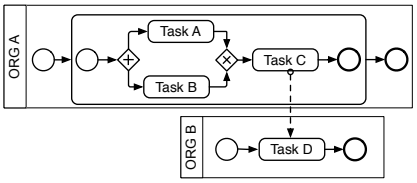


Fig. 9: Sound and message-aware unsound collaboration.

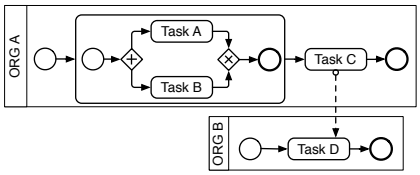


Fig. 10: Sound and message-aware sound collaboration.

and message-aware unsound. Now let consider another model obtained by Fig. 8 introducing a sub-process the resulting collaboration in Fig. 9 is message-aware unsound, since the use of the sub-process mitigates the causes. In fact, when the two processes reach a complete configuration, there will be a pending message, since task C sends two messages and only one will be consumed by task D. Differently, Fig. 10 shows that enclosing within a sub-process the part of the model generating multiple tokens it has also in this case a positive effect on the message-aware soundness of the model. The collaboration is message-aware sound, since only one message is sent and there are no pending messages. Soundness is also satisfied since it is implied by message-aware soundness.

4 Formal Framework

This section presents our formalisation of the BPMN semantics. Specifically, we first present the syntax and operational semantics we defined for a relevant subset of BPMN elements. The direct semantics proposed in this paper is inspired by [34], but its technical definition is significantly different. In particular, configuration states are here defined according to a global perspective, and the formalisation now includes choreography diagrams, which were overlooked in the previous semantics definition.

In selecting the elements we followed a pragmatic approach as, even if we deal with a restricted number of elements, we focus on those regularly used to design process models in practice (corresponding to less than 20% of the BPMN vocabulary [15]). Anyway, 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 [35]).

$$
\begin{array}{l}
C ::= \mathsf{pool}(\mathsf{p}, P) \quad | \quad C \parallel C \\
P ::= \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \mid \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \mid \mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}) \\
\quad \mid \mathsf{terminate}(\mathsf{e}_i) \mid \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh})) \\
\quad \mid \mathsf{andSplit}(\mathsf{e}_i, E_o) \mid \mathsf{xorSplit}(\mathsf{e}_i, E_o) \mid \mathsf{andJoin}(E_i, \mathsf{e}_o) \mid \mathsf{xorJoin}(E_i, \mathsf{e}_o) \\
\quad \mid \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o) \mid \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o) \\
\quad \mid \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \mid \mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o) \mid P \parallel P
\end{array}
$$

Fig. 11: Syntax of BPMN Collaboration Structures.

4.1 Syntax of BPMN Collaborations

To enable the formal treatment of collaborations' semantics, we defined a BNF syntax of their model structure (Fig. 11). In the proposed grammar, the non-terminal symbols C and P represent *Collaborations Structure* and *Processes Structure*, respectively. The two syntactic categories directly refer to the corresponding notions in BPMN. The terminal symbols, denoted by the sans serif font, are the typical elements of a BPMN model, i.e. pools, events, tasks, sub-processes and gateways.

It is worth noticing that our syntax is too permissive with respect to the BPMN notation, as it allows to write collaborations that cannot be expressed in BPMN. Limiting such expressive power would require to extend the syntax (e.g., by imposing processes having at least one end event), thus complicating the definition of the formal semantics. However, this is not necessary in our work, as 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.

Intuitively, a BPMN collaboration model is rendered in our syntax as a collection of pools and each pool contains a process. More formally, a Collaboration C is a composition, by means of operator $\parallel$ of pools of the form $\mathsf{pool}(\mathsf{p}, P)$, where: p is the name that uniquely identifies the Pool; P is the Process included in the specific pool, respectively.

In the following, $\mathsf{m} \in \mathbb{M}$ denotes a *message edge*, enabling message exchanges between pairs of participants in the collaboration, while $M \in 2^{\mathbb{M}}$. Moreover, m denote names uniquely identifying a message edge. We also observe $\mathsf{e} \in \mathbb{E}$ 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 events, i.e. $h > 1$ in each eventBased term. For the convenience of the reader, we refer with e_i 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 denoting the enabled status of start events and the complete status of end events, named *enabling* and *completing* edges, respectively. In particular, we use edge e_{enb}, incoming to a start event, to enable the activation of the process, while e_{cmp} is an edge outgoing from the end events suitable to check the completeness of the process. They are needed to activate sub-processes as well as to check their completion. Moreover, as well as for the message edge we have that e denotes names uniquely identifying a sequence edge.

The correspondence between the syntax used here to represent a *Process Structure* and the graphical notation of BPMN is as follows.

- $\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)$ represents a start event that can be activated by means of the enabling edge e_{enb} and it 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}.
- $\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o)$ represents a start message event that can be activated by means of the enabling edge e_{enb} as soon as a message m is received and it has an outgoing edge e_o.
- $\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp})$ represents an end message event with an incoming edge e_i, a message m to be sent, and a completing edge e_{cmp}.
- $\mathsf{terminate}(\mathsf{e}_i)$ represents a terminate event with an incoming edge e_i.
- $\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh}))$ represents an event based gateway with incoming edge e_i and a list of possible (at least two) message edges, with the related outgoing edges that are enabled by message reception.
- $\mathsf{andSplit}(\mathsf{e}_i, E_o)$ - 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.
- $\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)$ represents a task with incoming edge e_i and outgoing edge e_o; we can also observe $\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ - resp. $\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$ - to consider a task receiving - resp. sending - 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. sending - a message m.
- $\mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o)$ represents a sub-process element with incoming edge e_i and outgoing edge e_o. When activated, the enclosed sub-process P behaves according to the elements it consists of, including nested sub-process elements (used to describe collaborations with a hierarchical structure).
- $P_1 \| P_2$ represents a composition of elements in order to render a process structure in terms of a collection of elements.

Moreover, to simplify the definition of well-structured processes (given later), we include an $empty$ task in our syntax. It permits to connect two gateways with a sequence flow without activities in the middle.

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. To avoid malformed structure models, we only consider structures in which for each edge labeled by e (resp. m) outgoing from an element, there exists only one corresponding edge labeled by e (resp. m) incoming into another element, and vice versa.

Here in the following we define some auxiliary functions defined on the collaboration structure and the process structure. Considering BPMN collaborations they may include one or more participants; function $participant(C)$ returns the process structures

included in a given collaboration structure. Formally, it is defined as follows.

$$participant(C_1 \parallel C_2) = participant(C_1) \cup participant(C_1)$$
$$participant(\mathsf{pool}(\mathsf{p}, P)) = P$$

Since we also consider process including nested sub-processes to refer to the enabling edges of the start events of the current layer we resort to functions $start(P)$.

$$start(P_1 \parallel P_2) = start(P_1) \cup start(P_2)$$
$$start(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)) = \{\mathsf{e}_{enb}\} \qquad start(\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_{enb}\}$$
$$start(P) = \varnothing \text{ for any element } P \neq \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \text{ or } P \neq \mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o)$$

Notably, we assume that each process/sub-processes in the collaboration has only one start event. Function start applied to C will return as many enabling edges as the number of participants involved.

$$start(C_1 \parallel C_2) = start(participant(C_1)) \cup start(participant(C_2))$$
$$start(\mathsf{pool}(\mathsf{p}, P)) = start(P)$$

We similarly define functions $end(P)$ and $end(C)$ on the structure of processes and collaborations in order to refer to end events in the current layer.

$$end(P_1 \parallel P_2) = end(P_1) \cup end(P_2)$$
$$\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}) = \{\mathsf{e}_{cmp}\} \qquad end(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})) = \{\mathsf{e}_{cmp}\}$$
$$end(P) = \varnothing \text{ for any element } P \neq \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \text{ or } P \neq \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp})$$

The function $end(C)$ on the collaboration structure is defined as follow.

$$end(C_1 \parallel C_2) = end(participant(C_1)) \cup end(participant(C_2))$$
$$end(\mathsf{pool}(\mathsf{p}, P)) = end(P)$$

We also define the function $\mathsf{edges}(P)$ to refer the edges in the scope of P.

$$\mathsf{edges}(P_1 \parallel P_2) = \mathsf{edges}(P_1) \cup \mathsf{edges}(P_2)$$

$$\mathsf{edges}(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)) = \{\mathsf{e}_{enb}, \mathsf{e}_o\} \qquad \mathsf{edges}(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})) = \{\mathsf{e}_i, \mathsf{e}_{cmp}\}$$

$$\mathsf{edges}(\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_{enb}, \mathsf{e}_o\} \qquad \mathsf{edges}(\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp})) = \{\mathsf{e}_i, \mathsf{e}_{cmp}\}$$

$$\mathsf{edges}(\mathsf{terminate}(\mathsf{e}_i)) = \{\mathsf{e}_i\}$$

$$\mathsf{edges}(\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh}))) = \{\mathsf{e}_i, \mathsf{e}_{o1}, \ldots, \mathsf{e}_{oh}\}$$

$$\mathsf{edges}(\mathsf{andSplit}(\mathsf{e}_i, \mathsf{e}_{o1}, \ldots, \mathsf{e}_{oh})) = \{\mathsf{e}_i, \mathsf{e}_{o1}, \ldots, \mathsf{e}_{oh}\}$$

$$\mathsf{edges}(\mathsf{xorSplit}(\mathsf{e}_i, \mathsf{e}_{o1}, \ldots, \mathsf{e}_{oh})) = \{\mathsf{e}_i, \mathsf{e}_{o1}, \ldots, \mathsf{e}_{oh}\}$$

$$\mathsf{edges}(\mathsf{andJoin}(\mathsf{e}_{i1}, \ldots, \mathsf{e}_{ih}, , \mathsf{e}_o)) = \{\mathsf{e}_{i1}, \ldots, \mathsf{e}_{ih}, \mathsf{e}_o\}$$

$$\mathsf{edges}(\mathsf{xorJoin}(\mathsf{e}_{i1}, \ldots, \mathsf{e}_{ih}, \mathsf{e}_o)) = \{\mathsf{e}_{i1}, \ldots, \mathsf{e}_{ih}, \mathsf{e}_o\}$$

$$\mathsf{edges}(\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)) = \{\mathsf{e}_i, \mathsf{e}_o\}$$

$$\mathsf{edges}(\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_i, \mathsf{e}_o\} \qquad \mathsf{edges}(\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_i, \mathsf{e}_o\}$$

$$\mathsf{edges}(\mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o)) = \{\mathsf{e}_i, \mathsf{e}_o\} \qquad \mathsf{edges}(\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_i, \mathsf{e}_o\}$$

$$\mathsf{edges}(\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_i, \mathsf{e}_o\} \quad \mathsf{edges}(\mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o)) = \{\mathsf{e}_i, \mathsf{e}_o\} \cup \mathsf{edges}(P)$$

Running Example (2/9). The BPMN model in Fig. 1 is expressed in our syntax as the following collaboration structure (at an unspecified step of execution):

$$\mathsf{pool}(\mathsf{Customer}, P_C) \parallel \mathsf{pool}(\mathsf{TravelAgency}, P_{TA})$$

with P_C expressed as follows (process structure P_{TA} is defined in a similar way) where for simplicity we identify the edges in progressive order e_i (with $i = 0..10$):

$\mathsf{start}(\mathsf{e}_0, \mathsf{e}_1)$
$\parallel \mathsf{xorJoin}(\{\mathsf{e}_1, \mathsf{e}_3\}, \mathsf{e}_2) \parallel \mathsf{taskRcv}(\mathsf{e}_2, \mathsf{Offer}, \mathsf{e}_4) \parallel \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_3, \mathsf{e}_5\})$
$\parallel \mathsf{taskSnd}(\mathsf{e}_5, \mathsf{Travel}, \mathsf{e}_6) \parallel \mathsf{interRcv}(\mathsf{e}_6, \mathsf{Confirmation}, \mathsf{e}_7) \parallel \mathsf{taskSnd}(\mathsf{e}_7, \mathsf{Payment}, \mathsf{e}_8)$
$\parallel \mathsf{interRcv}(\mathsf{e}_8, \mathsf{Ticket}, \mathsf{e}_9) \parallel \mathsf{end}(\mathsf{e}_9, \mathsf{e}_{10})$

Moreover, considering functions we define on the structure we can observe the following: $participant(\mathsf{pool}(\mathsf{Customer}, P_C) \parallel \mathsf{pool}(\mathsf{TravelAgency}, P_{TA})) = \{P_C, P_{TA}\}$, $start(P_C) = \{\mathsf{e}_0\}$, and $end(P_C) = \{\mathsf{e}_{10}\}$. The others are defined in a similar way. □

Notice, the correspondence between the syntax used here to represent a BPMN model and the graphical notation of BPMN, that is exemplified by means of (an excerpt of) our running example in Fig. 1, is also reported in detail in the Appendix 11 considering the detailed one-to-one correspondence.

4.2 Semantics of BPMN Collaborations

The syntax presented so far permits to describe the mere structure of a collaboration and a process. 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* and *process configuration* these stateful descriptions.

Formally, a collaboration configuration has the form $\langle C, \sigma, \delta \rangle$, where: C is a collaboration structure; σ is the part of the execution state at process level, storing for each sequence edge the current number of tokens marking it (notice it refers to the edges included in all the process of the collaboration), and δ is the part of the execution state at collaboration level, storing for each message edge the current number of message tokens marking it. Moreover, a process configuration has the form $\langle P, \sigma \rangle$, where: P is a process structure; and σ is the execution state at process level. Specifically, a state $\sigma : \mathbb{E} \to \mathbb{N}$ is a function mapping edges to a number 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, a state $\delta : \mathbb{M} \to \mathbb{N}$ is a function mapping message edges to a number 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.

Given the notion of configuration, a collaboration is in the *initial state* when each process it includes is in the *initial state*, meaning that the start event of each process 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 nested sub-processes), and messages edges must be unmarked.

Definition 1 (Initial state of process). *Let $\langle P, \sigma \rangle$ be a process configuration. Predicate $isInit(P, \sigma)$ holds, if $\sigma(start(P)) = 1$, and $\forall\ \mathsf{e} \in \mathsf{edges}(P) \backslash start(P)\ .\ \sigma(\mathsf{e}) = 0$, then process configuration is initial if $isInit(P, \sigma)$ holds.*

Definition 2 (Initial state of collaboration). *Let $\langle C, \sigma, \delta \rangle$ be a collaboration configuration. Predicate $isInit(C, \sigma, \delta)$ holds, if $\forall\ P \in participant(C)$ we have that $isInit(P, \sigma)$, and $\forall\ \mathsf{m} \in \mathbb{M}\ .\ \delta(\mathsf{m}) = 0$, then a collaboration configuration is initial if $isInit(C, \sigma, \delta)$ holds.*

Running Example (3/9). The initial configuration of the collaboration in Fig. 1 is as follows. Given $participant(C) = \{P_C, P_{TA}\}$, we have that $\langle P_C, \sigma \rangle$, $\sigma(\mathsf{e}_0) = 1$ $\sigma(\mathsf{e}_i) = 0\ \ \forall \mathsf{e}_i$ with $i = 1..10$, and $\langle P_{TA}, \sigma \rangle$, $\sigma(\mathsf{e}_{11}) = 1$ and $\sigma(\mathsf{e}_j) = 0\ \ \forall \mathsf{e}_j$ with $j = 12..22$. We also have that $\delta(\mathsf{Offer}, \mathsf{Confirmation}, \mathsf{Ticket}, \mathsf{Travel}, \mathsf{Payment}) = 0$.

□

The operational semantics is defined by means of a *labelled transition system* (LTS) on collaboration configuration and formalises the execution of message marking evolution according to the process evolution. Its definition relies on an auxiliary transition relation on the behaviour of process.

The auxiliary transition relation is a triple $\langle \mathcal{P}, \mathcal{A}, \rightarrow \rangle$ where: $\mathcal{P}$, ranged over by $\langle P, \sigma \rangle$, is a set of process configurations; $\mathcal{A}$, ranged over by α, is a set of *labels* (of transitions that process configurations can perform); and $\rightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is a *transition relation*. We will write $\langle P, \sigma \rangle \xrightarrow{\alpha} \langle P, \sigma' \rangle$ to indicate that $(\langle P, \sigma \rangle, \alpha, \langle P, \sigma' \rangle) \in \rightarrow$ and say that process configuration $\langle P, \sigma \rangle$ performs a transition labelled by α to become process configuration $\langle P, \sigma' \rangle$. Since process execution only affects the current states,

and not the process structure, for the sake of readability we omit the structure from the target configuration of the transition. Thus, a transition $\langle P, \sigma \rangle \xrightarrow{\alpha} \langle P, \sigma' \rangle$ is written as $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$. The labels used by this transition relation are generated by the following production rules.

$$(Actions)\ \alpha ::= \tau \quad | \quad !\mathsf{m} \quad | \quad ?\mathsf{m} \qquad\qquad (Internal\ Actions)\ \tau ::= \epsilon \mid \ kill$$

The meaning of labels is as follows. Label τ denotes an action internal to the process, while $!\mathsf{m}$ and $?\mathsf{m}$ denote sending and receiving actions, respectively. The meaning of internal actions is as follows: ϵ the movement of a token through the process unless the termination action denoted by $kill$.

The transition relation over process configurations formalises the execution of a process; it is defined by the rules at the top of Fig. 12.

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 ways to sets of edges as follows: $inc(\sigma, \varnothing) = \sigma$ 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 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 ways to sets of edges as follows: $zero(\sigma, \varnothing) = \sigma$ and $zero(\sigma, \{\mathsf{e}\} \cup E)) = zero(zero(\sigma, \mathsf{e}), E)$.

To check the completion of a process and sub-process we exploit the boolean predicate $completed(P, \sigma)$. 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" [23, pp. 426, 431]. Thus, the process/sub-process completion can be formalised as follows.

Definition 3. *Let P be a process, having the form $\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \parallel P'$ or $\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}) \parallel P'$, the predicate $completed(P, \sigma)$ is defined as*

$$\sigma(\mathsf{e}_{cmp}) > 0 \wedge\ \sigma(\mathsf{e}_i) = 0 \wedge\ isZero(P', \sigma)$$

where $isZero(\cdot)$ is inductively defined on the structure of its first argument as follows:

- $isZero(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_{enb}) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$*;*
- $isZero(\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o))$ *if* $\sigma(\mathsf{e}_{enb}) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}))$ *if* $\sigma(\mathsf{e}_i) = 0$*;*
- $isZero(\mathsf{terminate}(\mathsf{e}_i), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$*;*
- $isZero(\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{oh})), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\forall j \in \{1, ..., h\}\ .\ \sigma(\mathsf{e}_{oj}) = 0$*;*

- $isZero(\mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\forall \mathsf{e} \in E_o \,.\, \sigma(\mathsf{e}) = 0$*;*
- $isZero(\mathsf{xorSplit}(\mathsf{e}_i, E_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\forall \mathsf{e} \in E_o \,.\, \sigma(\mathsf{e}) = 0$*;*
- $isZero(\mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma)$ *if* $\forall \mathsf{e} \in E_i \,.\, \sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{xorJoin}(E_i, \mathsf{e}_o), \sigma)$ *if* $\forall \mathsf{e} \in E_i \,.\, \sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma\rangle)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 0$*;*
- $isZero(\mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = \sigma(\mathsf{e}_o) = 0$ *and* $\forall \mathsf{e} \in \mathsf{edges}(P) \,.\, \sigma(\mathsf{e}) = 0$*;*
- $isZero(P_1 \| P_2, \sigma)$ *if* $isZero(P_1, \sigma)$ *and* $isZero(P_2, \sigma)$*.*

Notably, the completion of a process 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, which is defined as follows:

$$marked(\sigma, \{\mathsf{e}\} \cup E) = \begin{cases} \{\mathsf{e}\} \cup marked(\sigma, E) & \text{if } \sigma(\mathsf{e}) > 0; \\ marked(\sigma, E) & \text{otherwise.} \end{cases}$$

$marked(\sigma, \varnothing) = \varnothing$.

We now briefly comment on some of the operational rules in Fig. 12. Rule $P\text{-}Start$ starts the execution of a 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 $P\text{-}End$ 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 $P\text{-}StartRcv$ start the execution of a process when it is in its initial state. 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 enabling edge. A label corresponding to the consumption of a message is observed. Rule $P\text{-}EndSnd$ is enabled when there is at least a token in the incoming edge of the end event, which is then removed which is then moved to the completing edge. At the same time a label corresponding to the production of a message is observed. Rule $P\text{-}Terminate$ starts when there is at least one token in the incoming edge of the terminate event, which is then removed. Rule $P\text{-}EventG$ 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. A label corresponding to the consumption of a message is observed. Rule $P\text{-}AndSplit$ 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 $P\text{-}XorSplit$ 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 increment the token in one of the outgoing edges, non-deterministically chosen. Rule $P\text{-}AndJoin$ 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 $P\text{-}XorJoin$ is activated every time there is

a token in one of the incoming edges, which is then moved to the outgoing edge. Rule $P\text{-}Task$ 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 $P\text{-}TaskRcv$ is activated when there is a token in the incoming edge and a label corresponding to the consumption of a message is observed. Similarly, rule $P\text{-}TaskSnd$, instead of consuming, send a message before moving the token to the outgoing edge. A label corresponding to the production of a message is observed. Rule $P\text{-}InterRcv$ (resp. $P\text{-}InterSnd$) follows the same behaviour of rule $P\text{-}TaskRcv$ (resp. $P\text{-}TaskSnd$). Rule $P\text{-}Empty$ simply propagates tokens, it acts as a pass through. Rules $P\text{-}SubProcStart$, $P\text{-}SubProcEvolution$, $P\text{-}SubProcEnd$ and $P\text{-}SubProcKill$ deal with the behaviour of 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 according to the behaviour of the elements it contains according to the rules $P\text{-}SubProcEvolution$. When the sub-process completes the rule $P\text{-}SubProcEnd$ is activated. It removes all the tokens from the sequence edges of the sub-process body[1], and adds a token to the outgoing edge of the sub-process. Rule $P\text{-}SubProcKill$ deals with a sub-process element observing a killing action in its behaviour due to an occurrence of Terminate end event. This rule is activated only when there is a token in the incoming edge of termination event by the rule $P\text{-}Terminate$. Then, the sub-process stop its internal behaviours and passes the control to the upper layer, indeed the rule removes all the tokens in the sub-process and adds a token to the outgoing edge of the sub-process. Rule $P\text{-}Kill$ deal with the propagation of killing action on the scope of P and rule $P\text{-}Int$ deal with interleaving in a standard way for process elements. Notice that we do not need symmetric versions of the last two rules, as we identify processes up to commutativity and associativity of process collection.

[1] Actually, due to the completion definition, only the completing edges of the end events within the sub-process body need to be set to zero.

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

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

$$\langle \mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o), \sigma \rangle \xrightarrow{?\mathsf{m}} inc(dec(\sigma, \mathsf{e}_{enb}), \mathsf{e}_o) \quad \sigma(\mathsf{e}_{enb}) > 0 \quad (P\text{-}StartRcv)$$

$$\langle \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}), \sigma \rangle \xrightarrow{!\mathsf{m}} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_{cmp}) \quad \sigma(\mathsf{e}_i) > 0 \quad (P\text{-}EndSnd)$$

$$\langle \mathsf{terminate}(\mathsf{e}_i), \sigma \rangle \xrightarrow{kill} dec(\sigma, \mathsf{e}_i) \quad \sigma(\mathsf{e}_i) > 0 \quad (P\text{-}Terminate)$$

$$\langle \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh})), \sigma \rangle \xrightarrow{?\mathsf{m}_j} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_{oj}) \quad \sigma(\mathsf{e}_i) > 0, 1 \leqslant j \leqslant h \quad (P\text{-}EventG)$$

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

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

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

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

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

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

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

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

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

$$\langle \mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o), \sigma \rangle \xrightarrow{\epsilon} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o) \quad \sigma(\mathsf{e}_i) > 0 \quad (P\text{-}Empty)$$

$$\langle \mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o), \sigma \rangle \xrightarrow{\epsilon} inc(dec(\sigma, \mathsf{e}_i), start(P)) \quad \sigma(\mathsf{e}_i) > 0 \quad (P\text{-}SubProcStart)$$

$$\frac{\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'}{\langle \mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o), \sigma \rangle \xrightarrow{\alpha} \sigma'} \quad (P\text{-}SubProcEvolution)$$

$$\langle \mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o), \sigma \rangle \xrightarrow{\epsilon} inc(zero(\sigma, end(P)), \mathsf{e}_o) \quad completed(P, \sigma) \quad (P\text{-}SubProcEnd)$$

$$\frac{\langle P, \sigma \rangle \xrightarrow{kill} \sigma'}{\langle \mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o), \sigma \rangle \xrightarrow{kill} inc(zero(\sigma', \mathsf{edges}(P)), \mathsf{e}_o)} \quad (P\text{-}SubProcKill)$$

$$\frac{\langle P_1, \sigma \rangle \xrightarrow{kill} \sigma'}{\langle P_1 \parallel P_2, \sigma \rangle \xrightarrow{kill} zero(\sigma', \mathsf{edges}(P_1 \parallel P_2))} \quad (P\text{-}Kill) \qquad \frac{\langle P_1, \sigma \rangle \xrightarrow{\alpha} \sigma' \quad \alpha \neq kill}{\langle P_1 \parallel P_2, \sigma \rangle \xrightarrow{\alpha} \sigma'} \quad (P\text{-}Int)$$

$$\frac{\langle P, \sigma \rangle \xrightarrow{\tau} \sigma'}{\langle \mathsf{pool}(\mathsf{p}, P), \sigma, \delta \rangle \xrightarrow{\tau} \langle \sigma', \delta \rangle} \quad (C\text{-}Internal) \qquad \frac{\langle P, \sigma \rangle \xrightarrow{?\mathsf{m}} \sigma' \quad \delta(\mathsf{m}) > 0}{\langle \mathsf{pool}(\mathsf{p}, P), \sigma, \delta \rangle \xrightarrow{?\mathsf{m}} \langle \sigma', dec(\delta, \mathsf{m}) \rangle} \quad (C\text{-}Receive)$$

$$\frac{\langle P, \sigma \rangle \xrightarrow{!\mathsf{m}} \sigma'}{\langle \mathsf{pool}(\mathsf{p}, P), \sigma, \delta \rangle \xrightarrow{!\mathsf{m}} \langle \sigma', inc(\delta, \mathsf{m}) \rangle} \quad (C\text{-}Deliver) \qquad \frac{\langle C_1, \sigma, \delta \rangle \xrightarrow{\alpha} \langle \sigma', \delta' \rangle}{\langle C_1 \parallel C_2, \sigma, \delta \rangle \xrightarrow{\alpha} \langle \sigma', \delta' \rangle} \quad (C\text{-}Int)$$

Fig. 12: BPMN Semantics.

Now, the labelled transition relation on collaboration configurations formalises the execution of message marking evolution according to process evolution. In the case of collaborations, this is a triple $\langle \mathcal{C}, \mathcal{A}, \rightarrow \rangle$ where: $\mathcal{C}$, ranged over by $\langle C, \sigma, \delta \rangle$, is a set of collaboration configurations; $\mathcal{A}$, ranged over by α, is a set of *labels* (of transitions that collaboration configurations can perform as well as the process configuration); and $\rightarrow \subseteq \mathcal{C} \times \mathcal{A} \times \mathcal{C}$ is a *transition relation*. We will write $\langle C, \sigma, \delta \rangle \xrightarrow{\alpha} \langle C, \sigma', \delta' \rangle$ to indicate that $(\langle C, \sigma, \delta \rangle, \alpha, \langle C, \sigma', \delta' \rangle) \in \rightarrow$ and say that collaboration configuration $\langle C, \sigma, \delta \rangle$ performs transition labelled by α to become collaboration configuration $\langle C, \sigma', \delta' \rangle$. 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 $\langle C, \sigma, \delta \rangle \xrightarrow{\alpha} \langle C, \sigma', \delta' \rangle$ is written as $\langle C, \sigma, \delta \rangle \xrightarrow{\alpha} \langle \sigma', \delta' \rangle$. We recall α are the following: label τ denotes an action internal to the process, while $!\mathsf{m}$ and $?\mathsf{m}$ denote sending and receiving actions, respectively. The rules related to the collaboration are defined at the bottom of Fig. 12

The first three rules allow a single pool, representing organisation p, to evolve according to the evolution of its enclosed process P. In particular, if P performs an internal action, rule *C-Internal*, or a receiving/delivery action, rule *C-Receive*/*C-Deliver*, the pool performs the corresponding action at collaboration layer. Notably, rule *C-Receive* can be applied only if there is at least one message available (premise $\delta(\mathsf{m}) > 0$); of course, one token is consumed by this transition. Recall indeed that at process level label $?\mathsf{m}$ just indicates the willingness of a process to consume a received message, regardless the actual presence of messages. Moreover, when a process performs a sending action, represented by a transition labelled by $!\mathsf{m}$, such message is delivered to the receiving organization by applying rule *C-Deliver*. The resulting transition has the effect of increasing the number of tokens in the message edge m. The *C-Int* rule permits to interleave the execution of actions performed by pools of the same collaboration, so that if a part of a larger collaboration evolves, the whole collaboration evolves accordingly. Notice that we do not need symmetric versions of rule *C-Int*, as we identify collaborations up to commutativity and associativity of pools collection.

5 Properties of BPMN Collaborations

We provide here a rigorous characterisation, with respect to the BPMN formalisation presented so far, of the key concepts studied in this work: well-structuredness, safeness and soundness. We characterise these properties both at the level of processes and collaborations.

5.1 Well-Structured BPMN Collaborations

Common process modelling notations, such as BPMN, allow process models to have almost any topology. However, it is often desirable that models abide by some structural rules. In this respect, a well-known property of a process model is that of *well-structuredness*. In this paper we have been inspired by the definition of well-structuredness given by Kiepuszewski et. al. [17]. Such definition was given on workflow model and it is not expressive enough for BPMN, so we extend it to well-structuredness collaboration including all the elements defined in our semantics (i.e.

not only based element included in workflow model but also event based gateway and sub-processes).

Before providing a formal characterisation of well-structured BPMN processes and collaborations, we need to introduce some auxiliary definitions. In particular, we inductively define functions $in(P)$ and $out(P)$, which determine the incoming and outgoing sequence edges of a process element P as follows:

$$
\begin{array}{ll}
in(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)) = \varnothing & out(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})) = \{\mathsf{e}_i\} & out(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})) = \varnothing \\
in(\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o)) = \varnothing & out(\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp})) = \{\mathsf{e}_i\} & out(\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp})) = \varnothing \\
in(\mathsf{terminate}(\mathsf{e}_i)) = \{\mathsf{e}_i\} & out(\mathsf{terminate}(\mathsf{e}_i)) = \varnothing \\
in(\mathsf{andSplit}(\mathsf{e}_i, E_o)) = \{\mathsf{e}_i\} & out(\mathsf{andSplit}(\mathsf{e}_i, E_o)) = E_o \\
in(\mathsf{xorSplit}(\mathsf{e}_i, E_o)) = \{\mathsf{e}_i\} & out(\mathsf{xorSplit}(\mathsf{e}_i, E_o)) = E_o \\
in(\mathsf{andJoin}(E_i, \mathsf{e}_o)) = E_i & out(\mathsf{andJoin}(E_i, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{xorJoin}(E_i, \mathsf{e}_o)) = E_i & out(\mathsf{xorJoin}(E_i, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh}))) & out(\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h, \mathsf{e}_{oh}))) \\
\quad = \{\mathsf{e}_i\} & \quad = \{\mathsf{e}_{oj}\} \quad with \quad 1 < j < h \\
in(\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)) = \{\mathsf{e}_i\} & out(\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_i\} & out(\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_i\} & out(\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o)) = \{\mathsf{e}_i\} & out(\mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_i\} & out(\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_i\} & out(\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(\mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o)) = \{\mathsf{e}_i\} & out(\mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o)) = \{\mathsf{e}_o\} \\
in(P_1 \parallel P_2) = (in(P_1) \cup in(P_2)) & out(P_1 \parallel P_2) = (out(P_1) \cup out(P_2)) \\
\quad \setminus (out(P_1) \cup out(P_2)) & \quad \setminus (in(P_1) \cup in(P_2))
\end{array}
$$

Moreover, to simplify the definition of well-structuredness for processes, we also provide the definition of well-structured core by means of the boolean predicate $isWSCore(\cdot)$.

Definition 4 (Well-structured processes). *A process P is* well-structured *(WS) if P has one of the following forms:*

$$\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P' \parallel \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \quad (1)$$

$$\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P' \parallel \mathsf{terminate}(\mathsf{e}_i) \quad (2)$$

$$\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}) \quad (3)$$

$$\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \parallel P' \parallel \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \quad (4)$$

$$\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \parallel P' \parallel \mathsf{terminate}(\mathsf{e}_i) \quad (5)$$

$$\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \parallel P' \parallel \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}) \quad (6)$$

where $in(P') = \{\mathsf{e}_o\}$, $out(P') = \{\mathsf{e}_i\}$, *and* $isWSCore(P')$.
$isWSCore(\cdot)$ *is inductively defined on the structure of its first argument as follows:*

1. $isWSCore(\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o))$; *2.* $isWSCore(\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o))$;
3. $isWSCore(\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o))$; *4.* $isWSCore(\mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o))$;
5. $isWSCore(\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o))$; *6.* $isWSCore(\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o))$;

$$\frac{\forall j \in [1..n] \;\; isWSCore(P_j), \;\; in(P_j) \subseteq E_o, \;\; out(P_j) \subseteq E_i}{\begin{array}{l} 7. \;\; isWSCore(\mathsf{andSplit}(\mathsf{e}_i, E_o) \mid P_1 \mid \ldots \mid P_n \mid \mathsf{andJoin}(E_i, \mathsf{e}_o)) \\ 8. \;\; isWSCore(\mathsf{xorSplit}(\mathsf{e}_i, E_o) \mid P_1 \mid \ldots \mid P_n \mid \mathsf{xorJoin}(E_i, \mathsf{e}_o)) \end{array}}$$

$$9. \;\; \frac{\forall j \in [1..n] \;\; isWSCore(P_j), \;\; in(P_j) = \mathsf{e}_{oj}, \;\; out(P_j) \subseteq E_i}{isWSCore(\mathsf{eventBased}(\mathsf{e}_i, \{(\mathsf{m}_j, \mathsf{e}_{oj}) | j \in [1..n]\}) \mid P_1 \mid \ldots \mid P_n \mid \mathsf{xorJoin}(E_i, \mathsf{e}_o))}$$

$$10. \;\; \frac{isWSCore(P_1), isWSCore(P_2), in(P_1) = \{\mathsf{e}_1\}, out(P_1) = \{\mathsf{e}_4\}, in(P_2) = \{\mathsf{e}_6\}, out(P_2) = \{\mathsf{e}_2\}}{isWSCore(\mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1) \mid P_1 \mid P_2 \mid \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}))}$$

$$\frac{isWSCore(P_1')}{\begin{array}{l} 11(a). \;\; isWSCore(\mathsf{subProc}(\mathsf{e}_i, \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P_1' \parallel \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \mathsf{e}_o)) \\ 11(b). \;\; isWSCore(\mathsf{subProc}(\mathsf{e}_i, \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P_1' \parallel \mathsf{terminate}(\mathsf{e}_i), \mathsf{e}_o)) \\ 11(c). \;\; isWSCore(\mathsf{subProc}(\mathsf{e}_i, \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P_1' \parallel \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}), \mathsf{e}_o)) \\ 11(d). \;\; isWSCore(\mathsf{subProc}(\mathsf{e}_i, \mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \parallel P_1' \parallel \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \mathsf{e}_o)) \\ 11(e). \;\; isWSCore(\mathsf{subProc}(\mathsf{e}_i, \mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \parallel P_1' \parallel \mathsf{terminate}(\mathsf{e}_i), \mathsf{e}_o)) \\ 11(f). \;\; isWSCore(\mathsf{subProc}(\mathsf{e}_i, \mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \parallel P_1' \parallel \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}), \mathsf{e}_o)) \end{array}}$$

$$12. \;\; \frac{isWSCore(P_1), isWSCore(P_2), out(P_1) = in(P_2)}{isWSCore(P_1 \mid P_2)}$$

According to the definition 4, Well-structured processes is given in the forms (1-6) such as a possible combination of starting or end event according to the one included in the semantics. We allow a start event or a start message event and one end event such as simple end event or termination or end messege event. The start is connected to the body and the body to the end. In the middle the process body can be composed by any

element up to the well-structured core definition. Any single task or intermediate event is a well-structured core (cases 1-6); a composite node starting with an AND (resp. XOR, resp. Event-based) split and closing with an AND (resp. XOR, resp. XOR) join is well-structured core if each edge of the split is connected to a given edge of the join by means of a well-structured sub-node (cases 7-9); a loop of sequence edges ($e_1 \rightarrow e_4 \rightarrow e_6 \rightarrow e_2$) formed by means of a XOR split and a XOR join is well-structured core if the body of the loop consists of well-structured sub-nodes (case 10). Notably, only loops formed by XOR gateways are well-structured. For a better understanding, cases 7 - 10 are graphically depicted in Fig. 13. A subprocess is well structure core if it include

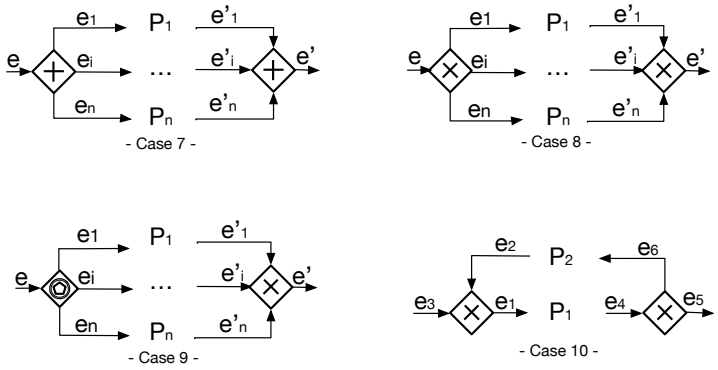


Fig. 13: Well-structured nodes (cases 7-10).

a well-structured process (case 11). A process element collection is well-structured core if its process are well-structured and sequentially composed (case 12).

Well-structuredness can be also simply extended to collaborations, by requiring each process involved in a collaboration to be well-structured.

Definition 5 (Well-structured collaborations). *Let C be a collaboration, $isWS(C)$ is inductively defined as follows:*

- $isWS(\mathsf{pool}(\mathsf{p}, P))$ *if P is well-structured;*
- $isWS(C_1 \parallel C_2)$ *if* $isWS(C_1)$ *and* $isWS(C_2)$.

Running Example (4/9). Considering the proposed running example and according to the above definitions, process P_C is well-structured, while process P_{TA} is not well-structured, due to the presence of the unstructured loop formed by the XOR join and an AND split. Thus, the overall collaboration is not well-structured. □

5.2 Safe BPMN Collaborations

Another important condition usually required is *safeness*, i.e the occurrence of no more than one token along the same sequence edge during process execution.

Before providing a formal characterisation of safeness BPMN processes and collaborations, we need to introduce the auxiliary function $maxMarking(\cdot)$ that, given a configuration $\langle P, \sigma \rangle$, determines the maximum number of tokens marking the sequence edges of elements in P according to the state σ (this function relies on the standard function $max(\cdot)$ returning the maximum in a list of natural numbers).

$$
\begin{aligned}
&maxMarking(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o), \sigma) = \sigma(\mathsf{e}_o)\\
&maxMarking(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma) = \sigma(\mathsf{e}_i)\\
&maxMarking(\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o), \sigma) = \sigma(\mathsf{e}_o)\\
&maxMarking(\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}), \sigma) = \sigma(\mathsf{e}_i)\\
&maxMarking(\mathsf{terminate}(\mathsf{e}_i), \sigma) = \sigma(\mathsf{e}_i)\\
&maxMarking(\mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma) = max(\sigma(\mathsf{e}_i), \sigma(E_o))\\
&maxMarking(\mathsf{xorSplit}(\mathsf{e}_i, E_o), \sigma) = max(\sigma(\mathsf{e}_i), \sigma(E_o))\\
&maxMarking(\mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma) = max(\sigma(E_i), \sigma(\mathsf{e}_o))\\
&maxMarking(\mathsf{xorJoin}(E_i, \mathsf{e}_o), \sigma) = max(\sigma(E_i), \sigma(\mathsf{e}_o))\\
&maxMarking(\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o), \sigma) = max(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o))\\
&maxMarking(\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma) = max(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o))\\
&maxMarking(\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma) = max(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o))\\
&maxMarking(\mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o), \sigma) = max(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o))\\
&maxMarking(\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma) = max(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o))\\
&maxMarking(\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma) = max(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o))\\
&maxMarking(\mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o), \sigma) = max(\sigma(\mathsf{e}_i), \sigma(\mathsf{edges}(P)), \sigma(\mathsf{e}_o))\\
&maxMarking(P_1 \parallel P_2, \sigma) = max(maxMarking(P_1, \sigma), maxMarking(P_2, \sigma))
\end{aligned}
$$

$maxMarking(\cdot)$ can be also simply extended to collaborations $\langle C, \sigma \rangle$, determines the maximum number of tokens marking the sequence edges of elements in all the process P included in the collaboration.

$$
\begin{aligned}
&maxMarking(\mathsf{pool}(\mathsf{p}, P), \sigma) = maxMarking(P, \sigma)\\
&maxMarking(C_1 \parallel C_2, \sigma) =\\
&\qquad max(maxMarking(participant(C_1), \sigma), maxMarking(participant(C_2), \sigma))
\end{aligned}
$$

We also need the following definition determining the safeness of a process in a given state.

Definition 6 (Current state safe process). *A process configuration $\langle P, \sigma \rangle$ is current state safe (*cs-safe*) if and only if $maxMarking(P, \sigma) \leqslant 1$.*

We can finally conclude with the definition of safe processes and collaborations which requires that cs-safeness is preserved along the computations. Now, a process is defined to be safe if it is preserved that the maximum marking does not exceed one along the process execution. We use $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$.

Definition 7 (Safe processes). *A process P is* safe *if and only if, given σ such that $isInit(P, \sigma)$, for all σ' such that $\langle P, \sigma \rangle \rightarrow^* \sigma'$ we have that $\langle P, \sigma' \rangle$ is cs-safe.*

Definition 8 (Safe collaborations). *A collaboration C is* safe *if and only if, given σ and δ such that $isInit(C, \sigma, \delta)$, for all for all σ' and δ' such that $\langle C, \sigma, \delta \rangle \rightarrow^* \langle \sigma', \delta' \rangle$ we have that $maxMarking(C, \sigma') \leqslant 1$.*

Running Example (5/9). Let us consider again our running example depicted in Fig. 1. Process P_C is safe since there is not any process fragment capable of producing more than one token. Process P_{TA} instead is not safe. In fact, if task Make Travel Offer is executed more than once, we would have that the AND split gateway will produce more than one token in the sequence flow connected to the Booking Received event. Thus, also the resulting collaboration is not safe. □

5.3 Sound BPMN Collaborations

Here we refer to the soundness as the need that from any reachable configuration it is possible to arrive in a (completed) configuration. This is possible under two different scenarios, the first one *(i)* where all marked end events are marked exactly by a single token and all sequence edges are unmarked, while the second *(ii)* when no token are observed in the configuration (i.e. the case of termination with terminate end event). This refers to the current state sound process we following define.

Definition 9 (Current state sound process). *A process configuration $\langle P, \sigma \rangle$ is current state sound (*cs-sound*) if and only if one of the following hold:*

(i) $\forall\, \mathsf{e}_{cmp} \in marked(\sigma, end(P))\ .\ \sigma(\mathsf{e}_{cmp}) = 1$ *and* $isZero(P, \sigma)$*;*
(ii) $\forall\, \mathsf{e} \in \mathsf{edges}(P)\ .\ \sigma(\mathsf{e}) = 0$.

Definition 10 (Sound process). *A process P is* sound *if and only if, given σ such that $isInit(P, \sigma)$, for all σ' such that $\langle P, \sigma \rangle \rightarrow^* \sigma'$ we have that there exists σ'' such that $\langle P, \sigma' \rangle \rightarrow^* \sigma''$, and $\langle P, \sigma'' \rangle$ is cs-sound.*

Definition 11 (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 $\langle C, \sigma', \delta' \rangle \rightarrow^* \langle \sigma'', \delta'' \rangle$, and $\forall\ P$ in C we have that $\langle P, \sigma'' \rangle$ is cs-sound.*

Thanks to the expressibility of our formalisation to distinguish sequence tokens from message tokens we provide a novel property, named message-aware soundness, that extend the usual soundness notion by considering sound those collaborations in which asynchronously sent messages are properlly handled by the receiver.

Definition 12 (Message-Aware 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 $\langle C, \sigma', \delta' \rangle \rightarrow^* \langle \sigma'', \delta'' \rangle$, and $\forall\ P$ in C we have that $\langle P, \sigma'' \rangle$ is cs-sound, and* $\forall\, \mathsf{m} \in \mathbb{M}\ .\ \delta''(\mathsf{m}) = 0$.

Running Example (6/9). Let us consider again our running example. It is easily to see that process P_C is sound, since it is always possible to reach the end event and when reached there is no token marking the sequence flows. Also process P_{TA} is sound, since when a token reaches the terminate event, all the other tokens are removed from the

edges by means of the killing effect. However, the resulting collaboration is not sound. In fact, when a travel offer is accepted by the customer, we would have that the AND-Split gateway will produce two tokens, one of which re-activates the task Make Travel Offer. Thus, even if the process completes, the message lists are not empty. However, the collaboration satisfied the Message-Disregarding sound property we define. □

6 Relationships among Properties

In this section we study the relationships among the considered properties. In particular we investigate the relationship between (i) well-structuredness and safeness, (ii) well-structuredness and soundness, and (iii) safeness and soundness. The proofs of these results are relegated to the Appendix 11.

6.1 Well-structuredness vs. Safeness in BPMN

In this section we present some of the main results of this work concerning the correlation between well-structuredness and safeness, both at process and collaboration level. Specifically, we demonstrate that all well-structured models are safe (Theorem 1), and that the vice versa does not hold. To this aim, first we show that a process in the initial state is cs-safe (Lemma 1). Then, we show that cs-safeness is preserved by the evolution of well-structured core process elements (Lemma 2) and processes (Lemma 3). These latter two lemmas rely on the notion of *reachable* processes. In fact, the syntax in Fig. 11 is too liberal, as it allows terms that cannot be obtained (by means of transitions) from a process in its initial state.

Definition 13 (Reachable processes). *A process configurations $\langle P, \sigma \rangle$ is* reachable *if there exists $\langle P, \sigma' \rangle$ configurations such that $isInit(P, \sigma')$ and $\langle P, \sigma' \rangle \rightarrow^* \sigma$.*

Lemma 1. *Let P be a process, if $isInit(P, \sigma)$ then $\langle P, \sigma \rangle$ is cs-safe.*

Proof (sketch). Trivially, from definition of $isInit(P, \sigma)$. □

Lemma 2. *Let $isWSCore(P)$, and let $\langle P, \sigma \rangle$ be reachable and cs-safe process configuration, if $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ then $\langle P, \sigma' \rangle$ is cs-safe.*

Proof (sketch). We proceed by induction on the structure of well-structured core process elements. □

Lemma 3. *Let P be WS, and let $\langle P, \sigma \rangle$ be a process configuration reachable and cs-safe, if $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ then $\langle P, \sigma' \rangle$ is cs-safe.*

Proof (sketch). We proceed by case analysis on the structure of P, which is a WS process (see Definition 4). □

Theorem 1. *Let P be a process, if P is well-structured then P is safe.*

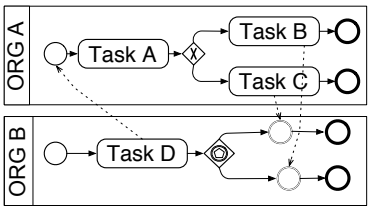


Fig. 14: A safe BPMN collaboration not well-structured

Proof (sketch). We show that if $\langle P, \sigma \rangle \rightarrow^* \sigma'$ then $\langle P, \sigma' \rangle$ is cs-safe, by induction on the length n of the sequence of transitions from $\langle P, \sigma \rangle$ to $\langle P, \sigma' \rangle$. □

The reverse implication of Theorem 1 is not true. In fact there are safe processes that are not well-structured. The collaboration diagram represented in Fig. 14 is an example. The involved processes are trivially safe, since there are not fragments capable of generating multiple tokens; however they are not well-structured.

We now extend the previous results to collaborations.

Theorem 2. *Let C be a collaboration, if C is well-structured then C is safe.*

Proof (sketch). We proceed by contradiction. □

6.2 Well-structuredness vs. Soundness in BPMN

In this section we present the relationship between well-structuredness and soundness, both at process and collaboration level. Specifically, we prove that a well-structured process is always sound (Theorem 3), but there are sound processes that are not well-structured. To this aim, first we show that a reachable well-structured core process element can always complete its execution (Lemma 4). This latter Lemma is based on the auxiliary definition of the final state of core elements in a process, given for all elements with the exception of start and end events.

Definition 14 (Final state of core elements in P.). *Let P be a process, then $isCompleteEl(P, \sigma)$ is inductively defined on the structure of process P as follows:*

$isCompleteEl(\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 1$
$isCompleteEl(\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 1$
$isCompleteEl(\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 1$
$isCompleteEl(\mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 1$
$isCompleteEl(\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 1$
$isCompleteEl(\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\sigma(\mathsf{e}_o) = 1$
$isCompleteEl(\mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\forall \mathsf{e} \in E_o \,.\, \sigma(\mathsf{e}) = 1$
$isCompleteEl(\mathsf{xorSplit}(\mathsf{e}_i, E_o), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$ *and* $\exists \mathsf{e} \in E_o \,.\, \sigma(\mathsf{e}) = 1$
and $\forall \mathsf{e}_k \in E_o \backslash \mathsf{e} \,.\, \sigma(\mathsf{e}) = 0$
$isCompleteEl(\mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma)$ *if* $\forall \mathsf{e} \in E_i \,.\, \sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}_o) = 1$
$isCompleteEl(\mathsf{xorJoin}(E_i, \mathsf{e}_o), \sigma)$ *if* $\forall \mathsf{e} \in E_i \,.\, \sigma(\mathsf{e}) = 0$ *and* $\sigma(\mathsf{e}_o) = 1$
$isCompleteEl(\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok})), \sigma)$ *if* $\sigma(\mathsf{e}_i) = 0$
and $\exists \mathsf{e} \in \{\mathsf{e}_{o1}, \ldots, \mathsf{e}_{ok}\} \,.\, \sigma(\mathsf{e}) = 1$ *and* $\forall \mathsf{e}_k \in \{\mathsf{e}_{o1}, \ldots, \mathsf{e}_{ok}\} \backslash \mathsf{e} \,.\, \sigma(\mathsf{e}) = 0$
$isCompleteEl(\mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o))$ *if* $\sigma(\mathsf{e}_i) = 0$, $\sigma(\mathsf{e}_o) = 1$ *and* $\forall \mathsf{e} \in \mathsf{edges}(P) \,.\, \sigma(\mathsf{e}) = 0$
$isCompleteEl(P_1 | P_2, \sigma)$ *if* $\forall \mathsf{e} \in out(P_1 \mid P_2) \,:\, isCompleteEl(\mathsf{getEl}(\mathsf{e}, P_1 \mid P_2))$
and $\forall \mathsf{e} \in (\mathsf{edges}(P_1 \mid P_2) \backslash out(P_1 \mid P_2)) \,:\, \sigma(\mathsf{e}) = 0$

where $\mathsf{getEl}(\mathsf{e}, P)$ *returns the element in* P *with incoming edge* e*:*

- $\mathsf{getEl}(\mathsf{e}, \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \begin{cases} \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) & \text{if } \mathsf{e} = \mathsf{e}_o \\ \epsilon & \text{otherwise} \end{cases}$
- $\mathsf{getEl}(\mathsf{e}, \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \begin{cases} \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) & \text{if } \mathsf{e} = \mathsf{e}_o \\ \epsilon & \text{otherwise} \end{cases}$
- $\mathsf{getEl}(\mathsf{e}, \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)) = \begin{cases} \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o) & \text{if } \mathsf{e} = \mathsf{e}_o \\ \epsilon & \text{otherwise} \end{cases}$
- $\mathsf{getEl}(\mathsf{e}, \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \begin{cases} \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) & \text{if } \mathsf{e} = \mathsf{e}_o \\ \epsilon & \text{otherwise} \end{cases}$
- $\mathsf{getEl}(\mathsf{e}, \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)) = \begin{cases} \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) & \text{if } \mathsf{e} = \mathsf{e}_o \\ \epsilon & \text{otherwise} \end{cases}$
- $\mathsf{getEl}(\mathsf{e}, \mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o)) = \begin{cases} \mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o) & \text{if } \mathsf{e} = \mathsf{e}_o \\ \epsilon & \text{otherwise} \end{cases}$
- $\mathsf{getEl}(\mathsf{e}, \mathsf{andSplit}(\mathsf{e}_i, E_o)) = \begin{cases} \mathsf{andSplit}(\mathsf{e}_i, E_o) & \text{if } \mathsf{e} \in E_o \\ \epsilon & \text{otherwise} \end{cases}$
- $\mathsf{getEl}(\mathsf{e}, \mathsf{andJoin}(E_i, \mathsf{e}_o)) = \begin{cases} \mathsf{andJoin}(E_i, \mathsf{e}_o) & \text{if } \mathsf{e} = \mathsf{e}_o \\ \epsilon & \text{otherwise} \end{cases}$
- $\mathsf{getEl}(\mathsf{e}, \mathsf{xorSplit}(\mathsf{e}_i, E_o)) = \begin{cases} \mathsf{xorSplit}(\mathsf{e}_i, E_o) & \text{if } \mathsf{e} \in E_o \\ \epsilon & \text{otherwise} \end{cases}$
- $\mathsf{getEl}(\mathsf{e}, \mathsf{xorJoin}(E_i, \mathsf{e}_o)) = \begin{cases} \mathsf{xorJoin}(E_i, \mathsf{e}_o) & \text{if } \mathsf{e} = \mathsf{e}_o \\ \epsilon & \text{otherwise} \end{cases}$

– $\mathsf{getEl}(\mathsf{e}, \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok}))) =$
$\begin{cases} \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_k, \mathsf{e}_{ok})) & \text{if } \mathsf{e} \in \{\mathsf{e}_{o1}, \ldots, \mathsf{e}_{ok}\} \\ \epsilon & \text{otherwise} \end{cases}$

– $\mathsf{getEl}(\mathsf{e}, P_1 \mid P_2) = \mathsf{getEl}(\mathsf{e}, P_1), \mathsf{getEl}(\mathsf{e}, P_2)$

Lemma 4. *Let $\langle P, \sigma \rangle$ be a reachable process configuration and $isWSCore(P)$, then there exists σ' such that $\langle P, \sigma \rangle \rightarrow^* \sigma'$ and $isCompleteEl(P, \sigma')$.*

Proof (sketch). We proceed by induction on the structure of well-structured core process. □

Theorem 3. *Let P be a WS process, then P is sound.*

Proof (sketch). We proceed by case analysis. □

The reverse implication of Theorem 3 is not true. In fact there are sound processes that are not well-structured; see for example the process represented in Fig. 15. This process is surely unstructured, and it is also trivially sound, since it is always possible to reach an end event without leaving tokens marking the sequence flows.

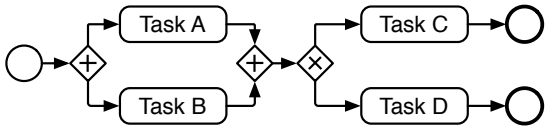


Fig. 15: An example of sound process not Well-Structured.

However, Theorem 3 does not extend to the collaboration level. In fact, when we put well-structured processes together in a collaboration, this could be either sound or unsound. This is also valid for message-disregarding soundness.

Theorem 4. *Let C be a collaboration, C is WS does not imply C is sound.*

Proof (sketch). We proceed by contradiction. □

Theorem 5. *Let C be a collaboration, C is WS does not imply C is message-aware sound.*

Proof (sketch). We proceed by contradiction. □

6.3 Safeness vs. Soundness in BPMN

In this section we present the relationship between safeness and soundness, both at process and collaboration level. Specifically we demonstrate that there are unsafe models that are sound. This is a peculiarity of BPMN, faithfully implemented in our semantics, thank to its capability to support the terminate end event and (unsafe) sub-processes. Let us first reason at process level considering some examples.

Example 1. Fig. 16 shows an example of unsafe process, since the AND split gateway produces two tokens that are then merged by the XOR join gateway producing two tokens on the outgoing edge of the XOR join. However, after Task C is executed and one token enables the terminate end event, the $kill$ label is produced and the second token in the sequence flow is removed (rule *P-Terminate*), rendering the process sound. □

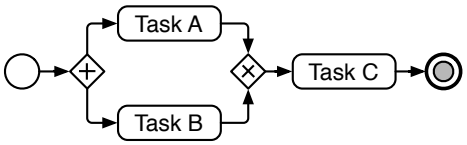


Fig. 16: An example of unsafe but sound process.

Theorem 6. *Let P be a process, P is unsafe does not imply P is unsound.*

Proof (sketch). We proceed by contradiction. □

Let us consider now the collaboration level. We have that unsafe collaborations could either sound or unsound, as proved by the following Theorem.

Theorem 7. *Let C be a collaboration, C is unsafe does not imply C is unsound.*

Proof (sketch). We proceed by contradiction. □

Running Example (9/9). Considering the collaboration in our running example, Customer is both safe and sound, while the process of the Travel Agency is unsafe but sound, since the terminate event permits a successfully termination of the process. The collaboration is not safe, and it is also sound but message-aware unsound, since there could be messages in the message lists.

Example 2. Let us consider the example in Fig. 17. The process in *ORG A* is unsafe but it is sound, since the terminate event permits a correct completion of the process. However, if the XOR split gateway of *ORG B* produces a token on the bottom sequence flow and Task E is executed, Task B will never received the message from Task D. Thus, even if each process has a token that reaches the terminate event and all the other tokens in the process are removed by the killing action, the message lists are not empty. Indeed, the collaboration is sound, but message-aware unsound. □

7 Properties Compositionality

In this section we study safeness and soundness compositionality, i.e. how the behaviour of processes affects that of the entire resulting collaboration. In particular, we show the interrelationship between the studied properties at collaboration and at process level. At process level we also consider the compositionality of sub-processes, investigating how sub-processes behaviour impacts on the safeness and soundness of process including them.

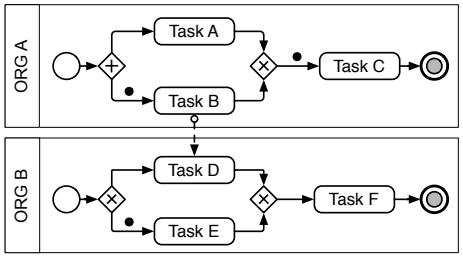


Fig. 17: An unsafe but sound collaboration.

7.1 On Safeness Compositionality

We show here that safeness is compositional, that is the composition of safe processes always results in a safe collaboration.

Theorem 8. *Let C be a collaboration, if all processes in C are safe then C is safe.*

Proof (sketch). We proceed by contradiction (see Appendix 11). □

We also show that the unsafeness of a collaboration cannot be in general determined by information about the unsafeness of the processes that compose it. Indeed, putting together an unsafe process with a safe or unsafe one, the obtained collaboration could be either safe or unsafe. Let us consider now some cases.

Running Example (7/9). In our example, the collaboration is composed by a safe process and an unsafe one. In fact, focussing on the process of the Travel Agency, we can immediately see that it is not safe: the loop given by a XOR join and an AND split produces multiple tokens on one of the outgoing edges of the AND split. Now, if we consider this process together with the safe process of Customer, the resulting collaboration is not safe. Indeed, the XOR split gateway, which checks if the offer is interesting, forwards only one token on one of the two paths. As soon as a received offer is considered interesting, the Customer process proceeds and completes. Thus, due to the lack of safeness, the travel agency will continue to make offers to the customer, but no more offer messages arriving from the Travel Agency will be considered by the customer. □

Example 3. Another example refers to the case in which a collaboration composed by a safe process and an unsafe one results in a safe collaboration, as shown in Fig. 18. If we focus only on the process in *ORG B* we can immediately notice that it is not safe: again the loop given by a XOR join and an AND split produces multiple tokens on the same edge. However, if we consider this process together with the safe process of *ORG A*, the resulting collaboration is safe. In fact, task D receives only one message, producing a token that is successively split by the AND gateway. No more message arrives from the send task, so, although there is a token is blocked, we have no problem of safeness. □

Example 4. In Fig. 19 we have two unsafe processes, since each of them contains a loop capable of generating an unbounded number of tokens. However, if we consider

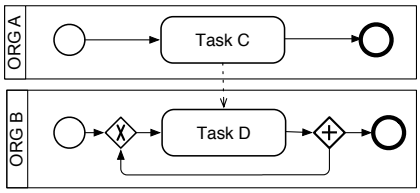


Fig. 18: Safe collaboration with safe and unsafe processes.

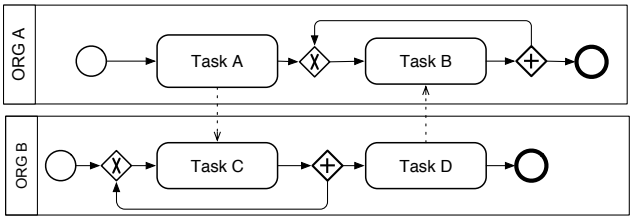


Fig. 19: Safe collaboration with unsafe processes.

the collaboration obtained by the combination of these processes, it turns out to be safe. Indeed, as in the previous example, tasks C and B are executed only once, as they receive only one message. Thus, the two loops are blocked and cannot effectively generate multiple tokens. □

Example 5. Also the collaboration in Fig. 20 is composed by two unsafe processes: process in *ORG A* contains an AND split followed by a XOR join that produces two tokens on the outgoing edge of the XOR gateway; process in *ORG B* contains the same loop as in the previous examples. In this case the collaboration composed by these two processes is unsafe. Indeed, the XOR join in *ORG A* will effectively produce two tokens since the sending of task B is not blocking. □

Let us now to consider processes including sub-processes. We show that the composition of unsafe sub-processes always results in un-safe processes, but the vice versa does not hold. There are also un-safe processes including safe sub-process when the unsafeness does not depend from the behaviour of the sub-process.

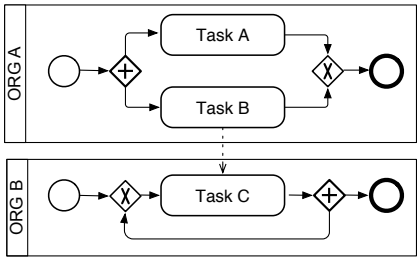


Fig. 20: Unsafe collaboration with unsafe processes.

Theorem 9. *Let P be a process including a sub-process* $\mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o)$*, if P_1 is unsafe then P is unsafe.*

Proof (sketch). We proceed by contradiction (see Appendix 11). □

7.2 On Soundness Compositionality

As well as for the safeness property, we show now that it is not feasible to detect the soundness of a collaboration by relying only on information about soundness of processes that compose it. However, the unsoundness of processes implies the unsoundness of the resulting collaboration.

Theorem 10. *Let C be a collaboration, if all processes in C are unsound then C is unsound.*

Proof (sketch). We proceed by contradiction (see Appendix 11). □

On the other hand, when we put together sound processes, the obtained collaboration could be either sound or unsound, since we have also to consider messages. It can happen that either a process waits for a message that will never be received or it receive more than the number of messages it is able to process. Let us consider some examples.

Running Example (8/9). In our running example, the collaboration is composed by two sound processes. In fact, the Customer process is well-structured, thus sound. Focussing on the process of the Travel Agency, it is also sound since when it completes the terminate end event aborts all the running activities and removes all the tokens still present (more details will follow in Section 3). However, the resulting collaboration is not message-aware sound, since the message lists could not be empty. □

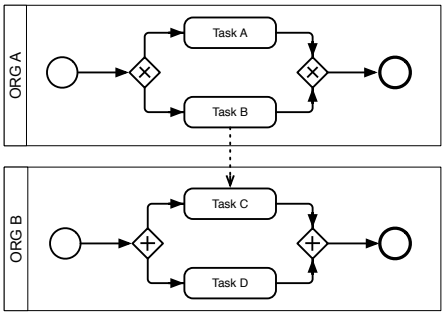


Fig. 21: An example of unsound collaboration with sound processes.

Example 6. In Fig. 21 we have a collaboration resulting from the composition of two sound processes. If we focus only on the processes in *ORG A* and *ORG B* we can immediately note that they are sound. However, the resulting collaboration is not sound.

In fact, for instance, if Task A is executed, Task C in *ORG B* will never receive the message and the AND-Join gateway cannot be activated, thus the process of *ORG B* cannot complete its execution. □

Example 7. Also the collaboration in Fig. 22 is trivially composed by two sound processes. However, in this case also the resulting collaboration is sound. In fact, Task E will always receive the message by Task B and the processes of *ORG A* and *ORG B* can correctly complete. □

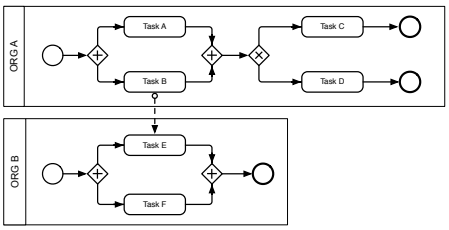


Fig. 22: Sound collaboration with sound processes.

Let's now to consider soundness in a multi-layer structure. We show that the composition of unsound sub-processes does not results in un-sound processes. There are also sound processes including unsound sub-process. In fact, when we put unsound sub-process together in a process, this could be either sound or unsound.

Theorem 11. *Let P be a process including a sub-process* $\mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o)$*, if P_1 is unsound does not imply P is unsound.*

Proof (sketch). We proceed by contradiction (see Appendix 11). □

8 Related Works

In this paper we provide a formal characterisation of well-structuredness BPMN models. To do that we have been inspired by the definition of well-structuredness given in [17]. Other attempts are also available in the literature. Van der Aalst et al. [36] state that a workflow net is well-structured if the split/join constructions are properly nested. El-Saber and Boronat [37] propose a formal definition of well-structured processes, in terms of a rewriting logic, but they do not extend this definition at collaboration level.

We than consider safeness, showing that this is a significant correctness property. Dijkman et al. [30] discuss about safeness in Petri Nets resulting from the translation of BPMN. In such work, safeness of BPMN terms means that no activity will ever be enabled or running more than once concurrently. This definition is given using the natural language, while in our work we give a precise characterisation of safeness for both BPMN processes and collaborations. Other approaches introducing mapping from

BPMN to formal languages, such as YAWL [38] and COWS [39], do not consider safeness, even if it is recognised as an important characteristic [40].

Moreover, soundness is considered as one of the most important correctness criteria. There is a jungle of definitions different notions of soundness in the literature, referring to different process languages and even for the same process language, e.g. for EPC soundness definition is given by Mendling in [41], and for Workflow Nets van der Aalst [19] provides two equivalent soundness definitions. However, these definitions cannot be used directly for BPMN because of its peculiarities. In fact, although the BPMN process flow resembles to some extent the behaviour of Petri Nets, it is not the same. BPMN 2.0 provides a comprehensive set of elements that go far beyond the definition of mere place/transition flows and enable modelling at an higher level of abstraction. For example, using Petri Nets it is difficult to describe certain operations typical of the business process domain, such as the termination event, and often it is required to rely on some limiting assumptions (e.g., safeness and well-structureness).

Other studies try to characterize inter-organizational soundness are available. A first attempt was done using a framework based on Petri Nets [20]. The authors investigate IO-soundness presenting an analysis technique to verify the correctness of an interorganizational workflow. However, the study is restricted to structured models. Soundness regarding collaborative processes is also given in [42] in the field of the Global Interaction Nets, in order to detect errors in technology-independent collaborative business processes models. However, differently from our work, this approach does not apply to BPMN, which is the modelling notation aimed by our study. Therefore, our investigation of properties at collaboration level provides novel insights with respect to the state-of-the-art of BPMN formal studies.

9 Relevance into Practice

To get a clearer idea of the impact of well-structuredness, safeness, and soundness on the real-world modelling practice, we have analyzed the BPMN 2.0 collaboration models available in a well-known, public, well-populated repository provided by the BPM Academic Initiative (`http://bpmai.org`). From the raw dataset, to avoid uncompleted models and low quality ones, we have selected only those with 100% of connectedness (i.e., all model elements are connected). This results on 2.740 models suitable for our investigation. To better understand the trend in Table 1, the models are grouped in terms of number of contained elements. From the technical point of view, well-structuredness has been checked using the PromniCAT platform[2], while safeness and soundness have been checked using the S^3 tool[3].

We have found that 86% of models in the repository are well-structured. Anyway, more interesting is the trend of the number of well-structured models with respect to their size. It shows that in practice BPMN models starts to become unstructured when their size grows. This means that structuredness should be regarded as a general guideline but one can deviate from it if necessary, especially in modelling complex scenarios. The balancing between the two classes motivates, on the one hand, our design choice

[2] `https://github.com/tobiashoppe/promnicat`
[3] `http://pros.unicam.it/s3/`

Size	Dataset	WS	Non-WS	Safe	MA-Sound	Sound
0 - 9	1668	1551(93%)	117(7%)	1647	1077	1133
10 - 19	910	692(76%)	218 (24%)	883	462	487
20 - 29	137	95(69%)	42(31%)	134	51	57
30 - 39	13	4 (27%)	9 (73%)	13	4	4
40 - 49	9	1(14%)	8 (86%)	9	3	3
50 - 59	1	0 (0%)	1 (100%)	1	0	0
60 - 69	0	0	0	0	0	0
70 - 79	2	0 (0%)	2 (100%)	2	0	0
0 - 79	2740	2342 (86%)	398 (14%)	2689	1597	1684

Table 1: Classification of the models in the BPM Academic Initiative repository.

of considering in our formalisation BPMN models with an arbitrary topology and, on the other hand, the necessity of studying well-structuredness and the related properties.

Concerning safeness, it results that 2.689 models are safe. The classes that surely cannot be neglected in our study, as they are suitable to model realistic scenarios, are those with size 20-29, 30-39 and 40-49 including 156 models, of which only 3 are unsafe. This makes evident that modelling safe models is part of the practice, and that imposing well-structuredness is sometimes too restrictive, since there is a huge class of models that are safe even if with an unstructured topology.

Concerning soundness, it results that there are 1.684 sound models. It results that modelling in a sound way is a common practice, recognizing soundness as one of the most important correctness criteria. Moreover, the data show that there are well-structured models that are not sound this confirm the limitation of well-structuredness. Concerning message-aware soundness, it results that the number of models satisfying this property is 87 less than the sound ones. This highlights the relevance of a set of models, up to now, not considered.

10 Concluding Remarks

Our study formally defines some important correctness properties, namely well-structuredness, safeness, and soundness, both at process and collaboration level. We demonstrate the relationships between the studied properties, with the aim of classifying BPMN collaboration diagrams according to the properties they satisfy. Rather than converting the BPMN model to a Petri or Workflow Net and studying relevant properties on the model resulting from the mapping we directly report such properties on BPMN considering its complexity. In doing this the approach is based an uniform formal framework and it is not limited to models of a specific topology, i.e., the models do not need to be block-structured.

Specifically, we show that well-structured collaborations represent a subclass of safe ones. In fact, there is a class of collaborations that are safe, even if with an unstructured topology. We also show there are well-structured collaborations that are neither sound nor message-aware sound. These models are typically discarded by the modelling approaches in the literature, as they are over suspected of carrying bugs. However, we

have shown that some of these models, hence they can play a significant role in practice. Finally, we demonstrate there are sound and message-aware sound collaborations that are not safe. Resulting classification provide a novel contribution by extending the reasoning from process to BPMN collaborations. Moreover, being close to the BPMN standard give use to catch the language peculiarities as the asynchronous communication models, and the completeness notion distinguishing the effect of end event and the terminate event.

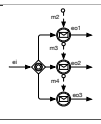
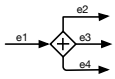
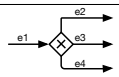
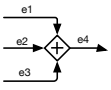
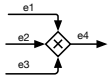
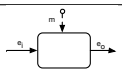
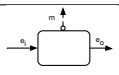
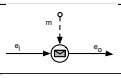
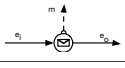
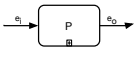
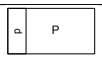
References

1. Lindsay, A., Downs, D., Lunn, K.: Business processes—attempts to find a definition. Information and Software Technology **45**(15) (2003) 1015–1019
2. OMG: Business Process Model and Notation (BPMN V 2.0) (2011)
3. Suchenia, A., Potempa, T., Ligeza, A., Jobczyk, K., Kluza, K.: Selected Approaches Towards Taxonomy of Business Process Anomalies. In: Advances in Business ICT: New Ideas from Ongoing Research. Volume 658 of SCI. Springer (2017) 65–85
4. 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
5. Dumas, M., La Rosa, M., Mendling, J., Mäesalu, R., Reijers, H.A., Semenenko, N.: Understanding business process models: the costs and benefits of structuredness. In: CAISE. Volume 7328 of LNCS. Springer (2012) 31–46
6. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Information Systems **37**(6) (2012) 518–538
7. Polyvyanyy, A., Garcia-Banuelos, L., Fahland, D., Weske, M.: Maximal Structuring of Acyclic Process Models. The Computer Journal **57**(1) (2014) 12–35
8. Polyvyanyy, A., Bussler, C.: The structured phase of concurrency. In: Information Systems Engineering. Springer (2013) 257–263
9. Corradini et al., F.: Classification of BPMN Collaborations. Tech.Rep., University of Camerino (2018) Available at: `http://pros.unicam.it/documents/`.
10. van der Aalst, W.M.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: Business Process Management, Models, Techniques, and Empirical Studies. Volume 1806 of LNCS. Springer (2000) 161–183
11. van der Aalst, W., van Hee, K., ter Hofstede, A., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.: Soundness of workflow nets: classification, decidability, and analysis. FAC **23**(3) (2011) 333–363
12. van der Aalst, W.M.: Process-oriented architectures for electronic commerce and interorganizational workflow. Information Systems **24**(8) (1999) 639–671
13. Murata, T.: Petri nets: Properties, analysis and applications. IEEE Proceedings **77**(4) (1989) 541–580
14. Rozenberg, G., Engelfriet, J.: Elementary net systems. In: Lectures on Petri Nets I: Basic Models. Springer (1998) 12–121
15. Muehlen, M.z., Recker, J.: How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: Advanced Information Systems Engineering. Volume 5074 of LNCS. Springer (2008) 465–479
16. Corradini, F., Ferrari, A., Fornari, F., Gnesi, S., Polini, A., Re, B., Spagnolo, G.O.: A guidelines framework for understandable BPMN models. Data Knowl. Eng. **113** (2018) 129–154
17. 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

18. van der Aalst, W.M.: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: Business Process Management, Models, Techniques, and Empirical Studies. Volume 1806 of LNCS. Springer (2000) 161–183
19. van der Aalst, W.M., van Hee, K.M., ter Hofstede, A.H., 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
20. van der Aalst, W.M.: Process-oriented architectures for electronic commerce and interorganizational workflow. Information Systems **24**(8) (December 1999) 639–671
21. El-Saber, N.A.: CMMI-CM compliance checking of formal BPMN models using Maude. PhD thesis, Department of Computer Science (2015)
22. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
23. OMG: Business Process Model and Notation (BPMN V 2.0) (2011)
24. Dehnert, J., Zimmermann, A.: On the suitability of correctness criteria for business process models. In: BPM. Volume 3649 of LNCS. Springer (2005) 386–391
25. van der Aalst, W.M.: Structural characterizations of sound workflow nets. Computing Science Reports **96**(23) (1996) 18–22
26. van Hee, K., Oanea, O., Serebrenik, A., Sidorova, N., Voorhoeve, M.: History-based joins: Semantics, soundness and implementation. In: International Conference on Business Process Management, Springer (2006) 225–240
27. Van der Aalst, W.M.: Verification of workflow nets. In: International Conference on Application and Theory of Petri Nets, Springer (1997) 407–426
28. Favre, C., Völzer, H.: Symbolic execution of acyclic workflow graphs. Business Process Management (2010) 260–275
29. Prinz, T.M.: Fast soundness verification of workflow graphs. In: ZEUS. Volume 1029 of LNCS. Springer (2013) 31–38
30. 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
31. Kunze, M., Weske, M.: Behavioural Models - From Modelling Finite Automata to Analysing Business Processes. Springer (2016)
32. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal verification of complex business processes based on high-level Petri nets. Information Sciences **385-386** (April 2017) 39–54
33. Ter Hofstede, A.: Workflow patterns: On the expressive power of (petri-net-based) workflow languages. PhD thesis, University of Aarhus (2002)
34. Corradini, F., Polini, A., Re, B., Tiezzi, F.: An Operational Semantics of BPMN Collaboration. In: FACS. Volume 9539 of LNCS., Springer (2015) 161 – 180
35. Corradini, F., Muzi, C., Re, B., Rossi, L., Tiezzi, F.: Global vs. Local Semantics of BPMN 2.0 OR-Join. In: 44th International Conference on Current Trends in Theory and Practice of Computer Science. Volume 10706 of LNCS. Springer (2018) 321–336
36. Van Der Aalst, W.M.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems and Computers **08**(01) (1998) 21–66
37. El-Saber, N., Boronat, A.: BPMN Formalization and Verification Using Maude. In: Workshop on Behaviour Modelling-Foundations and Applications, ACM (2014) 1–12
38. Decker, G., Dijkman, R., Dumas, M., García-Bañuelos, L.: Transforming BPMN diagrams into YAWL nets. In: BPM. Volume 5240 of LNCS. Springer (2008) 386–389
39. Prandi, D., Quaglia, P., Zannone, N.: Formal Analysis of BPMN Via a Translation into COWS. In: Coordination Models and Languages. Volume 5052 of LNCS. Springer (2008) 249–263
40. Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. In: Foundations of Software Technology and Theoretical Computer Science. Volume 761 of LNCS. Springer (1993) 326–337

41. Mendling, J.: Detection and prediction of errors in EPC business process models. PhD thesis, Wirtschaftsuniversität Wien Vienna (2007)
42. Roa, J., Chiotti, O., Villarreal, P.: A verification method for collaborative business processes. In: International Conference on Business Process Management. Volume 99 of LNBIP. Springer (2011) 293–305

11 Appendix: Correspondence

Graphical Representation	Textual Notation
	$\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)$
	$\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$
	$\mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o)$
	$\mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp})$
	$\mathsf{terminate}(\mathsf{e}_i)$
	$\mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), (\mathsf{m}_2, \mathsf{e}_{o2}), (\mathsf{m}_3, \mathsf{e}_{o3}))$
	$\mathsf{andSplit}(\mathsf{e}_1, \{\mathsf{e}_2, \mathsf{e}_3, \mathsf{e}_4\})$
	$\mathsf{xorSplit}(\mathsf{e}_1, \{\mathsf{e}_2, \mathsf{e}_3, \mathsf{e}_4\})$
	$\mathsf{andJoin}(\{\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_4)$
	$\mathsf{xorJoin}(\{\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_4)$
	$\mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)$
	$\mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$
	$\mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$
	$\mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o)$
	$\mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$
	$\mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o)$
	$\mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o)$
	$\mathsf{pool}(\mathsf{p}, P)$

Here we reported the complete correspondence between the BPMN graphical notation and our syntax. For the sake of presentation, join and split gateways include only three incoming/outgoing branching respectively.

Appendix: Proofs

In this appendix we report the proofs of the results presented in the paper.

Lemma 1 *Let P be a process, if $isInit(P, \sigma)$ then $\langle P, \sigma \rangle$ is cs-safe.*

Proof. Trivially, from definition of $isInit(P, \sigma)$. By definition of $isInit(P, \sigma)$, we have that $\sigma(\mathsf{e}_{enb}) = 1$ where $\mathsf{e}_{enb} \in start(P)$ and $\forall\ \mathsf{e} \in \mathbb{E} \backslash start(P)\ .\ \sigma(\mathsf{e}) = 0$, i.e. only the start event has a marking and all the other edges are unmarked. Hence, we have that $maxMarking(P, \sigma) \leqslant 1$, which allows us to conclude. □

Lemma 2 *Let $isWSCore(P)$, and let $\langle P, \sigma \rangle$ be reachable and cs-safe process configuration, if $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ then $\langle P, \sigma' \rangle$ is cs-safe.*

Proof. We proceed by induction on the structure of WSCore process elements.
Base cases: we show here only few interesting cases among the multiple base cases; since by hypothesis N is WS, it can only be either a task or an intermediate event. Let us consider the simple task, since all the other cases are similar. □

- $P = \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)$. By hypothesis $\langle P, \sigma \rangle$ is cs-safe, then $maxMarking(P, \sigma) = \max(\sigma(\mathsf{e}_i), \sigma(\mathsf{e}_o)) \leqslant 1$. The only rule that can be applied to infer the transition $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ is $P\text{-}Task$. In order to apply the rule there must be $0 < \sigma(\mathsf{e}_i)$; hence $0 < \sigma(\mathsf{e}_i) \leqslant 1$, i.e. $\sigma(\mathsf{e}_i) = 1$. We can exploit the fact that $\langle P, \sigma \rangle$ be is a reachable process configuration to prove that $\sigma(\mathsf{e}_o) = 0$. The application of the rule produces $\sigma' = \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o) \rangle$, i.e. $\sigma(\mathsf{e}_i) = 0$ and $\sigma(\mathsf{e}_o) = 1$. Thus, $maxMarking(P, \sigma') = \sigma(\mathsf{e}_o)$. Since $\sigma(\mathsf{e}_o) = 1$ we have that $maxMarking(P, \sigma') \leqslant 1$, which allows us to conclude.

Inductive cases: we consider the following cases, the other are deal with similarly.

- Let us consider $\langle \mathsf{andSplit}(\mathsf{e}_i, E_o) \mid P_1 \mid \ldots \mid P_n \mid \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma \rangle$. There are the following possibilities:
  - $\langle \mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma \rangle$ evolves by means of rule $P\text{-}AndSplit$. We can exploit the fact that this is a reachable well-structured process configuration to prove that $\sigma(\mathsf{e}_i) = 1$ and $\forall \mathsf{e} \in E_o\ .\sigma(\mathsf{e}) = 0$. Thus, $\langle \mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}_i), E_o) \rangle$. Hence, $maxMarking(\mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma') = 1$. By hypostesis $\langle \mathsf{andSplit}(\mathsf{e}_i, E_o) \mid P_1 \mid \ldots \mid P_n \mid \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma \rangle$ is cs-safe, i.e. if there is a token on the state $\langle \mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma' \rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term is cs-safe.
  - Node $P_1 \mid \ldots \mid P_n$ evolves without affecting the split and join gateways. In this case we can easily conclude by inductive hypothesis.
  - Node $P_1 \mid \ldots \mid P_n$ evolves and affects the split and/or join gateways. In this case we can reason like in the first case, by relying on inductive hypothesis.
  - $\langle \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma \rangle$ evolves by means of rule $P\text{-}AndJoin$. We can exploit the fact that this is a reachable well-structure node to prove that $\forall \mathsf{e} \in E_i\ .\sigma(\mathsf{e}) = 1$ and $\sigma(\mathsf{e}_o) = 0$. Thus $\langle \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, E_i), \mathsf{e}_o) \rangle$. Hence, $maxMarking(\mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma') = 1$. By hypothesis $\langle \mathsf{andJoin}(E_i, \mathsf{e}_o) \mid P_1 \mid \ldots \mid P_n \mid \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma \rangle$ is cs-safe, i.e. if there is a token on the state $\langle \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma' \rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term is cs-safe.
- Let us consider $\mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1) \mid P_1 \mid P_2 \mid \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\})$ with $in(P_1) = \{\mathsf{e}_1\}$, $out(P_1) = \{\mathsf{e}_4\}$, $in(P_2) = \{\mathsf{e}_6\}$, $out(P_2) = \{\mathsf{e}_2\}$

- $\langle \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1), \sigma \rangle$ evolves by means of rule *P-XorJoin*. We can exploit the fact that this is a reachable well-structured process configuration to prove that the term is marked $\sigma(\mathsf{e}_1) = 0$ and either $\sigma(\mathsf{e}_2) = 1$ or $\sigma(\mathsf{e}_3) = 1$; let us assume the marking is $\sigma(\mathsf{e}_3) = 1$ (since the other case is similar). Thus $\langle \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}_2), \mathsf{e}_1) \rangle$. Hence, $maxMarking(\mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1), \sigma' \rangle) = 1$. By hypothesis $\langle \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1)\sigma \rangle$ is cs-safe, i.e. if there is a token on the state $\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E_i, \mathsf{e}_o), \sigma' \rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term is cs-safe.
- Node $P_1 \mid P_2$ evolves without affecting the split and join gateways. In this case we can easily conclude by inductive hypothesis.
- Node $P_1 \mid P_2$ evolves and affects the split xor join and xor split gateways. In this case we can reason like in the first case, by relying on inductive hypothesis.
- $\langle \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}), \sigma \rangle$ evolves by means of rule *P-XorSplit*. We can exploit the fact that this is a reachable well-structure node to prove that the term is marked either as $\sigma(\mathsf{e}_4) = 1$. Hence, it evolves in a cs-safe term; in fact let us assume that it evolves in this way $\langle \mathsf{xorSplit}(\mathsf{e}_i, \{\mathsf{e}\} \cup E_o), \sigma \rangle \xrightarrow{\mathsf{e}_i} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}) \rangle$. Hence, $maxMarking(\mathsf{xorSplit}(\mathsf{e}_i, \{\mathsf{e}\} \cup E_o), \sigma') = 1$. By hypostesis $\langle \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_3\}, \mathsf{e}_1) \mid P_1 \mid P_2 \mid \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}), \sigma \rangle$ is cs-safe, i.e. if there is a token on the state $\langle \mathsf{xorSplit}(\mathsf{e}_4, \{\mathsf{e}_5, \mathsf{e}_6\}), \sigma' \rangle$ all the other edges do not have token. This means that cs-safeness is not affected. Therefore, the overall term is cs-safe.

– Let us consider $\langle P, \sigma \rangle = \langle P_1 \mid P_2, \sigma \rangle$. The relevant case for cs-safeness is when P evolves by applying *P-Int₁*. We have that $\langle P_1 \mid P_2, \sigma \rangle \xrightarrow{\alpha} \sigma'$ with $\langle P_1, \sigma \rangle \xrightarrow{\alpha} \sigma'$. By definition of maxMarking function we have that $maxMarking(P, \sigma)$ = $\max(maxMarking(P_1, \sigma), maxMarking(P_2, \sigma))$. By inductive hypothesis we have that $maxMarking(P_1, \sigma) = maxMarking(P_1, \sigma') \leqslant 1$ which is cs-safe. Since P_2 is well structured and cs-safe, then also $\langle P_2, \sigma' \rangle$ is cs-safe, which permits us to conclude.

□

Lemma 3 *Let P be WS, and let $\langle P, \sigma \rangle$ be a process configuration reachable and cs-safe, if $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ then $\langle P, \sigma' \rangle$ is cs-safe.*

Proof. According to Definition 4, P can have 6 different forms. We proceed by case analysis on the parallel component of $\langle P, \sigma \rangle$ that causes the transition $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$.

We show now the case P= $\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P' \parallel \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$.

– $\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o)$ evolves by means of the rule *P-Start*. In order to apply the rule there must be $\sigma(\mathsf{e}_{enbl}) > 0$, hence, by cs-safeness, $\sigma(\mathsf{e}_{enbl}) = 1$. We can exploit the fact that this is a reachable well-structured configuration to prove that $\sigma(\mathsf{e}_o) = 0$. The rule produces the following transition $\langle \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}_{enb}), \mathsf{e}_o) \rangle$ where $\sigma(\mathsf{e}_{enb}) = 0$ and $\sigma(\mathsf{e}_o) = 1$. Now, $\langle P, \sigma \rangle = \langle \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P' \parallel \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma \rangle$ can evolve only through the application of *P-Int₁* producing $\langle P, \sigma' \rangle$ with $\sigma(in(P')) = 1$.
By hypothesis $\langle P, \sigma \rangle$ is cs-safe, thus $\sigma(\mathsf{e}_i) \leqslant 1$, $\sigma(\mathsf{e}_{cmp}) \leqslant 1$ and $\max(\sigma(\mathsf{edges}(P'))) \leqslant 1$. Now $maxMarking(P', \sigma) \leqslant 1$ and $maxMarking(P', \sigma') \leqslant 1$. Therefore $maxMarking(P, \sigma') = max(0, 1, \sigma(in(P')), \sigma(out(P')), \sigma(\mathsf{e}_i), \sigma(\mathsf{e}_{cmp})) \leqslant 1$, then $\langle P, \sigma' \rangle$ is cs-safe.
– $\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$ evolves by means of the rule *P-End*. We can exploit the fact that this is a reachable well-structured configuration to prove that the term is marked as $\sigma(\mathsf{e}_i) = 1$ and $\sigma(\mathsf{e}_{cmp}) = 0$. The rule produces the following transition $\langle \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_{cmp}) \rangle$. Now, $\langle P, \sigma \rangle$ can only evolve by applying *P-Int₁* producing $\langle P, \sigma' \rangle$.

By hypothesis $\langle P, \sigma \rangle$ is cs-safe, then $\sigma(\mathsf{e}_i) \leqslant 1$, $\sigma(\mathsf{e}_{cmp}) \leqslant 1$ and P' is cs-safe. Reasoning as previously we can conclude that $\langle P, \sigma' \rangle$ is cs-safe.

– P' moves, that is $\langle P', \sigma \rangle \xrightarrow{\alpha} \sigma'$. By Lemma 2 $\langle P', \sigma' \rangle$ is safe, thus $maxMarking(P', \sigma') \leqslant 1$. By hypothesis, P is cs-safe therefore $maxMarking(\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o), \sigma') \leqslant 1$, $maxMarking(\mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma') \leqslant 1$. We can conclude that $\langle P, \sigma' \rangle$ is safe.

Now we consider the case P= $\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P' \parallel \mathsf{terminate}(\mathsf{e}_i)$.

– The start event evolves: like the previous case.
– The end terminate event evolves: the only transition we can apply is *P-Terminate*. By applying the rule we have$\langle \mathsf{terminate}(\mathsf{e}_i), \sigma \rangle \xrightarrow{kill} dec(\sigma, \mathsf{e}_i)$ with $\sigma(\mathsf{e}_i) > 0$. Now, $\langle P, \sigma \rangle$ can only evolve by applying $P\text{-}Kill_1$ producing $\langle P, \sigma' \rangle$ where σ' is completed unmarked; therefore it is cs-safe.
– P' moves: similar to the previous case.

□

Theorem 1 *Let P be a process, if P is well-structured then P is safe.*

Proof. We have to show that if $\langle P, \sigma \rangle \rightarrow^* \sigma'$ then $\langle P, \sigma' \rangle$ is cs-safe. We proceed by induction on the length n of the sequence of transitions from $\langle P, \sigma \rangle$ to $\langle P, \sigma' \rangle$.
Base Case ($n = 0$): In this case $\sigma = \sigma'$, then $isInit(P, \sigma')$ is satisfied. By Lemma 1 we conclude $\langle P, \sigma' \rangle$ is cs-safe.
Inductive Case: In this case $\langle P, \sigma \rangle \rightarrow^* \langle P, \sigma'' \rangle \xrightarrow{\alpha} \langle P, \sigma' \rangle$ for some process $\langle P, \sigma'' \rangle$. By induction, $\langle P, \sigma'' \rangle$ is cs-safe. By applying Lemma 3 to $\langle P, \sigma'' \rangle \xrightarrow{\alpha} \langle P, \sigma' \rangle$, we conclude $\langle P, \sigma' \rangle$ is cs-safe. □

Theorem 2 *Let C be a collaboration, if C is well-structured then C is safe.*

Proof. By contradiction, let us assume C is well-structured and C is unsafe. By Definition 8, there exists a collaboration configuration $\langle C, \sigma', \delta' \rangle$ such that $\langle C, \sigma, \delta \rangle \rightarrow^* \langle C, \sigma', \delta' \rangle$ and $maxMarking(C, \sigma') > 1$ and $\langle P, \sigma' \rangle$ not cs-safe. Thus, there exists P in C such that $\langle P, \sigma \rangle \rightarrow^* \langle P, \sigma' \rangle$. From hypothesis $isInit(C, \delta)$, we have $isInit(P, \sigma)$. From hypothesis C is well-structured, we have that P is WS. Therefore, by Theorem 1, P is safe. By Definition 7, $\langle P, \sigma' \rangle$ is cs-safe, which is a contradiction. □

Lemma 4 *Let $\langle P, \sigma \rangle$ be a reachable process configuration and $isWSCore(P)$, then there exists σ' such that $\langle P, \sigma \rangle \rightarrow^* \sigma'$ and $isCompleteEl(P, \sigma')$.*

Proof. We proceed by induction on the structure of $isWSCore(P)$. Base cases: by definition of $isWSCore()$, P can only be either a task or an intermediate event; we show here only the case in which it is a non communicating task, the other are dealt with similarly.

– $P = \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o)$. The only rule we can apply is *P-Task*. In order to apply the rule there must be $\sigma(\mathsf{e}_i) > 0$. Since $isWSCore(P)$, $\langle P, \sigma \rangle$ is safe, hence $\sigma(\mathsf{e}_i) = 1$. Since the process configuration is reachable we have $\sigma(\mathsf{e}_o) = 0$. The application of the rule produces $\langle \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o) \rangle$. Thus, we have $\sigma(\mathsf{e}_i) = 0$ and $\sigma(\mathsf{e}_o) = 1$, which permits us to conclude.

Inductive cases: we consider one case, the other are dealt with similarly.

– Let us consider $P = \langle \mathsf{andSplit}(\mathsf{e}_i, E_o) \mid P_1 \mid \ldots \mid P_n \mid \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma \rangle$. There are the following possibilities:

- $\langle \mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma \rangle$ evolves by means of rule $P\text{-}AndSplit$. We can exploit the fact that this is a reachable well-structured process configuration to prove that $\sigma(\mathsf{e}_i) = 1$ and $\forall \mathsf{e} \in E_o \ .\sigma(\mathsf{e}) = 0$. Thus, $\langle \mathsf{andSplit}(\mathsf{e}_i, E_o), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}_i), E_o) \rangle$. Now, P can evolve only through the application of $P\text{-}Int_1$ producing $\langle P, \sigma_1' \rangle$ with $\sigma_1'(in(P_1)) = \ldots = \sigma''(in(P_n)) = 1$. By inductive hypothesis there exists a state σ_1' such that $isCompleteEl(P_1 \mid \ldots \mid P_n, \sigma_1')$. Now, P can only evolve by applying rule $P\text{-}Int_1$, producing $\langle P, \sigma_2' \rangle$ with $\sigma_2'(\mathsf{edges}(E_i)) = 1$. Now, $\langle \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma_2' \rangle$ can evolve by means of rule $P\text{-}AndJoin$. The application of the rule produces $\langle \mathsf{andJoin}(E_i, \mathsf{e}_o), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, E_i), \mathsf{e}_o) \rangle$, i.e. $\sigma(\mathsf{e}_o) = 1$ and $\forall \mathsf{e} \in E_i \ .\sigma(\mathsf{e}) = 0$. This permits us to conclude.
- $P_1 \mid \ldots \mid P_n$ evolves without affecting the split and join gateways. In this case we can easily conclude by inductive hypothesis.
- $P_1 \mid \ldots \mid P_n$ evolves and affects the split and/or join gateways. In this case we can reason like in the first case

□

Theorem 3 *Let $\langle P, \sigma \rangle$ be a WS process configuration, then $\langle P, \sigma \rangle$ is sound.*

Proof. According to Definition 4, P can have 6 different forms. We consider now the case P= $\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P' \parallel \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp})$.

Let us assume that $isInit(P, \sigma)$. Thus we have that $\sigma(start(P)) = 1$, and $\forall \ \mathsf{e} \in \mathsf{edges}(P) \backslash start(P) \ . \ \sigma(\mathsf{e}) = 0$. Therefore the only parallel component of P that can infer a transition is the start event. In this case we can apply only the rule $P\text{-}Start$. The rule produces the following transition, $\langle \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o), \sigma \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}_{enb}), \mathsf{e}_o) \rangle$ where $\sigma(\mathsf{e}_{enb}) = 0$ and $\sigma(\mathsf{e}_o) = 1$. Now $\langle P, \sigma \rangle$ can evolve through the application of rule $P\text{-}Int_1$ producing $\langle P, \sigma_1' \rangle$, with $\sigma_1'(in(P')) = 1$. Now P' moves. By hypothesis $isWSCore(P')$, thus by Lemma 4 there exists a process configuration $\langle P', \sigma_2' \rangle$ such that $\langle P, \sigma \rangle \rightarrow^* \sigma_2'$ and $isCompleteEl(P', \sigma_2')$. The process can now evolve thorough rule $P\text{-}Int_1$. By hypothesis the process is WS, thus, after the application of the rule we obtain $\langle \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \parallel P' \parallel \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}), \sigma_3' \rangle$, where $\sigma_3'(\mathsf{e}_i) = 1$ and $\forall \mathsf{e} \in \mathsf{edges}(P') \ . \ \sigma_3'(\mathsf{e}) = 0$. We can now apply rule $P\text{-}End$ that decrements the token in e_i and produces a token in e_{cmp}, which permits us to conclude. □

Theorem 4 *Let C be a collaboration, C is WS does not imply C is sound.*

Proof. Let C be a WS collaboration, and let us suppose that C is sound. Then, it is sufficient to show a counter example, i.e. a WS collaboration that is not sound. Let us consider, for instance, the collaboration in Fig. 23. By Definition, the collaboration is WS. The soundness of the collaboration instead depends on the evaluation of the condition of the XOR-Split gateway in ORG A. If a token is produced on the upper flow and Task A is executed then Task C in ORG B will never receive the message and the AND-Join gateway can not be activated, thus the process of ORG B can not complete its execution. □

Theorem 5 *Let C be a collaboration, C is WS does not imply C is message-aware sound.*

Proof. Let C be a WS collaboration, and let us suppose that C is message-aware sound. Then, it is sufficient to show a counter example, i.e. a WS collaboration that is not message-aware sound. We can consider again the collaboration in Fig. 23. By reasoning as previously, the message-aware soundness of the collaboration depends on the evaluation of the condition of the XOR-Split gateway in ORG A. This permits us to conclude. □

Theorem 6 *Let C be a collaboration, C is unsafe does not imply C is unsound.*

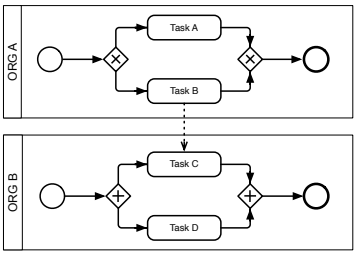


Fig. 23: An example of unsound collaboration with sound WS processes.

Proof. Let C be a unsafe collaboration, and let us suppose that C is unsound. Then, it is sufficient to show a counter example, i.e. a unsafe collaboration that is sound. We can consider the collaboration in Fig. 24. Process in ORG A and ORG B are trivially unsafe, since the AND split gateway produces two tokens that are then merged by the XOR join gateway producing two tokens on the outgoing edge of the XOR join. By definition of safeness collaboration the considered collaboration is unsafe. Concerning soundness, processes of ORG B and ORG A are sound. In fact, in each process, after one token enables the terminate end event, the kill label is produced and the second token in the sequence flow is removed (rule P-Terminate), permits the successfully termination of the collaboration. Thus, the resulting collaboration is sound. □

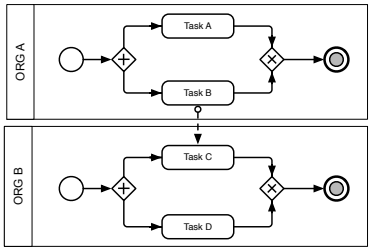


Fig. 24: An example of unsafe but sound collaboration.

Theorem 7 *Let C be a collaboration, if all processes in C are safe then C is safe.*

Proof. By contradiction let C be unsafe, i.e. there exists a collaboration $\langle C, \sigma', \delta' \rangle$ such that $\langle C, \sigma, \delta \rangle \rightarrow^* \langle \sigma', \delta' \rangle$ with $\mathsf{pool}(\mathsf{p}, P)$ in C and $\langle P, \sigma' \rangle$ not cs-safe. By hypothesis all processes of C are safe, hence it is safe the process, say P, of organisation p. As $\langle C, \sigma', \delta' \rangle$ results from the evolution of $\langle C, \sigma, \delta \rangle$, the process $\langle P, \sigma' \rangle$ must derive from $\langle P, \sigma \rangle$ as well, that is $\langle P, \sigma \rangle \rightarrow^* \sigma'$. By safeness of P, we have that $\langle P, \sigma' \rangle$ is cs-safe, which is a contradiction. □

Theorem 8 *Let P be a process including a sub-process $\mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o)$, if P_1 is unsafe then P is unsafe.*

Proof. Let us suppose $P = \mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o) \parallel P_2$ By contradiction let P be safe, i.e. given σ such that $isInit(P, \sigma)$, for all σ' such that $\langle P, \sigma \rangle \rightarrow^* \sigma'$ we have that $\langle P, \sigma' \rangle$ is cs-safe. By hypothesis P_1 is unsafe, i.e. given σ_1' such that $isInit(P_1, \sigma_1')$, there exists σ_2'

such that $\langle P_1, \sigma_1' \rangle \rightarrow^* \sigma_2'$ and $\langle P_1, \sigma_2' \rangle$ not cs-safe. Thus, we have $maxMarking(P_1, \sigma_2') \geqslant 1$. By definition of function $maxMarking()$, we have that $maxMarking(P, \sigma_2') = \max(maxMarking(\mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o)), maxMarking(P_2)) = maxMarking(P_1, \sigma_2') \geqslant 1$. Thus, P is not cs-safe, which is a contradiction. □

Theorem 9 *Let C be a collaboration, if all processes in C are unsound then C is unsound.*

Proof. Let P_1 and P_2 be two unsound processes and let C be the collaboration obtained putting together P_1 and P_2. By contradiction let C be sound, i.e., 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 $\langle C, \sigma', \delta' \rangle \rightarrow^* \langle \sigma'', \delta'' \rangle$, and $\forall\ P$ in C we have that $\langle P, \sigma'' \rangle$ is cs-sound. Since P_1 and P_2 are unsound, we have, for instance, that, given σ_1', such that $isInit(P_1, \sigma_1')$, for all σ_2' such that $\langle P, \sigma \rangle \rightarrow^* \sigma_2'$ we have that there not exists σ_3' such that $\langle P, \sigma_2' \rangle \rightarrow^* \sigma_3'$, and $\langle P, \sigma_3' \rangle$ is cs-sound. Choosing $\langle C, \sigma', \delta' \rangle$ such that $\mathsf{pool}(\mathsf{p}, P_1)$ in C', by unsoundness of P_1 we have that there exists a process in C' that is not cs-sound, which is a contradiction. □

Theorem 10 *Let P be a process including a sub-process $\mathsf{subProc}(\mathsf{e}_i, P_1, \mathsf{e}_o)$, if P_1 is unsound does not imply P is unsound.*

Proof. Let P_1 be a unsound, and let us suppose that P is unsound. Then, it is sufficient to show a counter example, i.e. an sound process including an unsound sub-process. We can consider process in Fig. 25. The process is unsound since when there is a token in the end event of ORG A there is still a pending sequence token to be consumed. If we include the part of the model generating multiple tokens in the scope of a sub-process, as it is shown in Fig. 26, that is when the process includes a sub-process, the process is sound. In fact, when there is a token in the end event of ORG A no other pending sequence token need to be processed. □

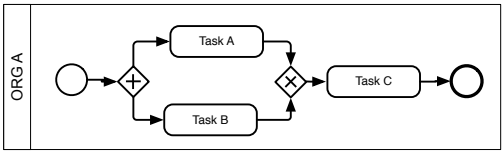


Fig. 25: An example of unsound process.

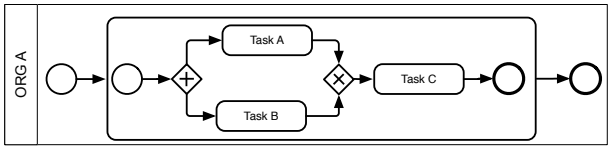


Fig. 26: An example of sound process with unsound sub-processes.