A Classification of BPMN Collaborations based on Safeness and Soundness Notions

Flavio Corradini, Chiara Muzi, Barbara Re, Francesco Tiezzi

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

{*name.surname*}@unicam.it

BPMN 2.0 standard has a huge uptake in modelling business processes within the same organisation or collaborations involving multiple interacting participants. 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 in the business process domain, 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

Modern organisations recognise the importance of having tools to describe how to behave in order to reach their own 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. [. . .] Business processes 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*" [22]. Up to now, several languages have been proposed to represent business process models. The OMG standard BPMN [27] is the most prominent language, since it is widely accepted in both academia and industry. In particular, BPMN collaboration models are used to describe distributed and complex scenarios where multiple participants interact via the exchange of messages.

Eventhough widely accepted, BPMN's major drawback is related to the possible misunderstanding of its execution semantics, defined by means of natural text descriptions, sometimes containing misleading information [33]. To overcome this issue, much effort has been devoted to formalise BPMN semantics by means of mapping it to other formal languages. The most relevant is the one to Petri Nets provided by Dijkman et al. [13]. However, models resulting from a mapping inherit constraints given by the target formal language and so far none of them considers BPMN features such as the different abstraction levels (i.e., collaboration, process, and sub-process), the asynchronous communication model, and the notion of termination due to different types of end 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 collaboration models according to relevant properties of the business process domain. It is worth noticing that our work aims at providing a classification specific for the BPMN notation. To this aim, our formal semantics is directly

defined on BPMN elements. Our intention is to introduce a unique formal framework to allow BPMN designers to achieve a better understanding of their models, and relative properties. This results in a systematic methodological approach to improve the design of BPMN collaborations.

As a distinctive aspect, the proposed semantics supports models with arbitrary topology, including of course also 'well-structured' ones [21]. This is necessary to enable a classification of both structured and unstructured models with respect to specific properties. Unstructured models can often be studied by means of structured ones at the cost of considerable increasing their size [14], altough this is not always possible [28, 29]. Looking at the public repository of BPMN models provided by the BPM Academic Initiative (`http://bpmai.org`) we noticed that unstructured models are largely used in practice, especially when the size of the models is significant [9].

Regarding the considered properties, our classification relies on a well-known class of properties in the domain of business process management, namely *safeness* [6] and *soundness* [7, 4]. So far, no formal definition of such properties directly given on BPMN is provided, despite the large body of work on this topic. We reconcile in our single framework properties taken into account by different languages, like Petri Nets [26], Workflow Nets [6], and Elementary Nets [32]. Studying different properties in the same framework does not leave any room for ambiguity, and increases the potential for formal reasoning on BPMN. Differently from other formal notations, our framework primitively allows to express important features of the BPMN notation, such as message passing and relative soundness properties (e.g. message-relaxed soundness).

Hence, the main contribution of the paper is a classification of BPMN collaborations according to relevant properties of the domain. More in detail, we prove that a well-structured collaboration is always safe, but the reverse does not hold. Moreover, well-structuredness implies soundness only at the process level, while there are well-structured collaborations that are not sound. Regarding the relationships between soundness and safeness, we prove that soundness does not imply safeness. Indeed, there are unsafe models that are sound. Similarly, sound models are not necessarily safe. Moreover, we study safeness and soundness compositionality in the domain of business process management, and show how specific BPMN constructs, namely terminate event and sub-processes, move certain BPMN models from one class to another. To illustrate both our formal framework and properties, as well as their relationships, we rely on a running example concerning a travel agency scenario.

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 it clear the relationships between them. Sec. 7 presents the study on safeness and soundness compositionality. Finally, Sec. 8 discusses related works, and Sec. 9 concludes the paper.

2   Background Notions

In this section we first provide some basic notions on elements that can be included in BPMN collaboration diagrams. Then, we present a travel agency collaboration scenario, used throughout the paper as a running example. Finally, we illustrate the properties we considered in the classification.

Basic Notions on BPMN. Here we do not aim at providing a complete presentation of the standard, but a discussion of the main concepts of BPMN [27] we use in the following. Our choice of the considered BPMN fragment is driven by practical aspects. Indeed, as shown in [25], even if the BPMN specification is quite wide, only less than 20% of its vocabulary is used regularly in designing business process models.

The BPMN elements we considere are those constructs that are most used in practice. They are
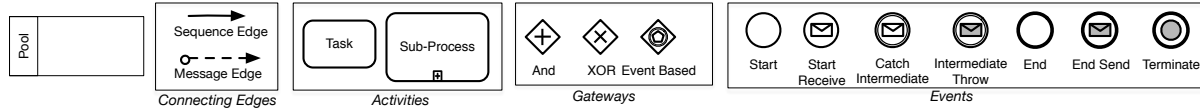
Figure 1: Considered BPMN 2.0 Elements.

briefly described below and reported in Fig. 1. **Pools** are used to represent participants or organisations involved in a collaboration, and include details on internal process specifications and related elements. **Connecting Edges** are used to connect process elements in the same or different pools. *Sequence Edge* is a solid connector used to specify the internal flow of the process, thus ordering elements in the same pool, while *Message Edge* is a dashed connector used to visualise communication flows between organisations. **Activities** are used to represent specific works to be performed within a process. In particular, a *task* is an atomic activity. It represents work that can not interrupted. It can be also used to send and receive messages. A *sub-process* represents work that is broken down to a finer level of detail. **Gateways** are used to manage the control flow of a process both for parallel activities and choices. Gateways act as either join nodes (merging incoming sequence edges) or split nodes (forking into outgoing sequence edges). Different types of gateways are available. A *XOR* gateway gives the possibility to model choices. In particular, a XOR-Split gateway is used after a decision to fork the flow into branches. When executed, it activates exactly one outgoing edge. A XOR-Join gateway acts as a pass-through, meaning that it is activated each time the gateway is reached. An *AND* gateway enables parallel execution flows. An AND-Split gateway is used to model the parallel execution of two or more branches, as all outgoing sequence edges are activated simultaneously. An AND-Join gateway synchronises the execution of two or more parallel branches, as it waits for all incoming sequence edges to complete before triggering the outgoing flow. An *Event-Based* is used after a decision to fork the flow into branches according to external choice. Its outgoing branches activation depends on taking place of catching events. Basically, such events are in a race condition, where the first event that is triggered wins and disables the other ones. **Events** are used to represent something that can happen. An event can be a *Start Event* representing the point from which a process starts, an *Intermediate Event* representing something that happens during process execution, or an *End Event* representing the process termination. Events are drawn as circles. When an event is source or target of a message edge, it is called *Message Event*. According to the different kinds of message edge connections, we distinguish between: (i) *Start Message Event* is a start event with an incoming message edge; the event element catches a message and starts a process; (ii) *Throw Intermediate Event* is an intermediate event with an outgoing message edge; the event element sends a message; (iii) *Catch Intermediate Event* is an intermediate event with an incoming message edge; the event element receives a message, (iv) *End Message Event* is an end event with an outgoing message edge; the event element sends a message and ends the process. We also refer to a particular type of end event, the *Terminate End Event* able to stop and abort the running process. Finally, a key concept related to the BPMN process execution refers to the notion of *token*. The BPMN standard states that "a token is a theoretical concept that is used as an aid to define the behaviour of a process that is being performed" [27, Sec. 7.1.1]. A token is commonly generated by a start event, traverses the sequence edges of the process and passes through its elements enabling their execution, and it is consumed by an end event when process completes. The distribution of tokens in the process elements is called *marking*, therefore the *process execution* is defined in terms of marking evolution. In the collaboration, the process execution also triggers message flow able to genere messages. We will refer them as message flow token.

Travel Agency Collaboration Scenario. Considering some of the introduced BPMN elements we can obtain the travel agency scenario in Fig. 2, which combines in a collaboration model the activities of a Customer and a Travel Agency. In particular, the Travel Agency continuously offers travels to the
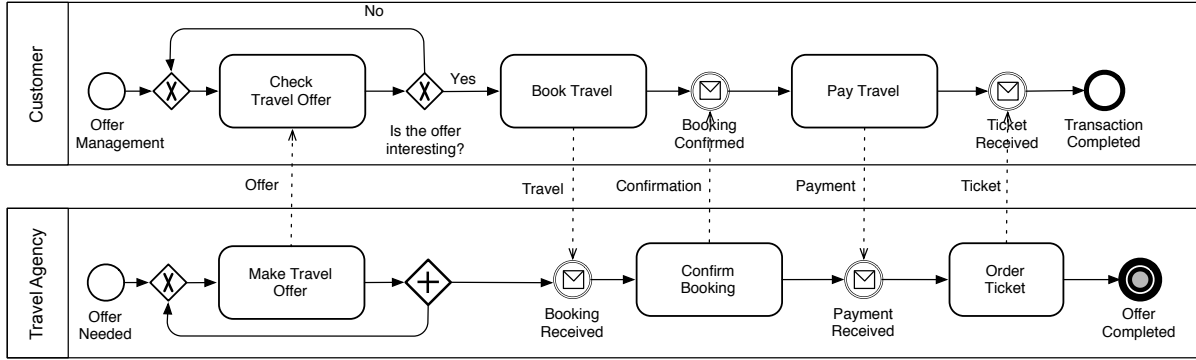
Figure 2: BPMN collaboration diagram of a Travel Agency scenario.

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 end. The processes of the Customer and of the Travel Agency are represented inside two *pools*. Considering the Customer pool, from left to right, we have that as soon as the process starts, due to its *start event*, the Customer checks a received travel offer. This is done by executing a *receiving task*. Then, she decides either to book the travel or to wait for other offers, by means of a loop composed of two *gateways*: a XOR-Join and a XOR-Split. After the Customer finds the interesting offer, she books the travel, by sending a message to the Travel Agency via a *sending task*, and waits for the booking confirmation. As soon as she receives the booking confirmation, through an *intermediate receiving event*, she pays the travel, receives the ticket from the Travel Agency and her specific works terminate by means of an *end event*. 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*. Thanks to the looping behaviour obtained by an AND-Split combined with a XOR-Join, it continuously makes 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 event*, which stops and aborts the running process, including the offering of travels.

Well-structuredness, Safeness and Soundness for BPMN. We introduce now an informal definition of *well-structuredness*, *safeness* and *soundness* for BPMN models, while their formalisation is provided in Section 5. In particular, well-structuredness relates to the way the various model elements are connected with each other, while safeness and soundness relate to the way a process model can be executed.

A BPMN process model is *well-structured* [21], if 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. The notion of well-structuredness can be extended to process collaborations (see Def. 1), requiring that the processes of all involved organisations are well-structured.

A BPMN process model is *safe*[1], if during its execution no more than one token occurs along the same sequence edge (see Def. 3). This property is inspired by the Petri-Net literature, where safeness means that a Petri Net does not contain more than one token in all reachable markings [6]. Safeness naturally extends to process collaborations, requiring that the processes of all involved organisations

---

[1]Notably, the notion of safeness is different from that of safety, since it is a standard term in the BPM literature.

satisfy it considering the overall collaboration execution (see Def. 4).

A BPMN process model is *sound*, if it can complete its execution without leaving active elements and all the model elements can be activated in at least one of the execution traces (see Def. 6). This property is inspired by the business process literature that, since the mid nineties, presents several versions of soundness considering different modelling languages [6, 7, 4, 16]. To escape from this jungle of definitions and focussing on the specificities of BPMN, here we based on the classical soundness definition for processes [15] 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. Considering the BPMN notion of completion [27, pp. 426, 431], requiring that all tokens in that instance must reach an end node, the *no dead activities* property is equivalent to requiring the complete execution of each process instance. 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 completion, while the second case is not caught by our semantics since reasoning on models with data is out of the scope of this work. Finally, we extend the soundness property to process collaborations (see Def. 7).

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 [12, 1] reasoning only at the process level and considering the Workflow Nets semantics, our study directly addresses BPMN collaboration models. The obtained results we formally prove are synthesised in the Euler diagram in Fig. 3, showing in particular that:

(i) all well-structured collaborations are safe, but the reverse does not hold;

(ii) there are well-structured collaborations that are neither sound nor message-relaxed sound;

(iii) there are sound and message-relaxed sound collaborations that are not safe.
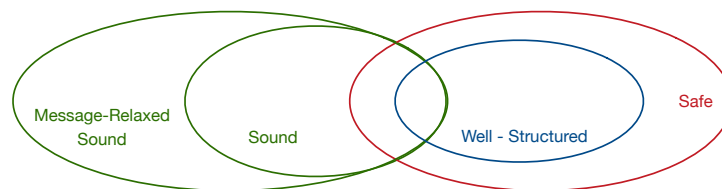

Figure 3: Classification of BPMN collaborations.

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 a relation is valid at the 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 [1]. 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 [18], and relaxing the one obtained in Petri Nets [12], where relaxed soundness and well-structuredness together imply soundness. Results *(i)* and *(ii)* together confirm limits of well-structuredness as a correctness criterion. Indeed, considering only well-structuredness is too strict, as some safe and sound models that are not

well-structured result discarded right from the start. Result *(iii)* shows that there are sound and message-relaxed sound collaborations that are not safe. This also can be observed at the process level resulting in 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 [2]. 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 [17, 30]. Thus, a sound workflow is always safe. In BPMN instead there are unsafe processes that are sound. Summing up, our results are novel and also influence the reasoning at process level. This is mainly due to the effects of the terminate event and sub-process behaviour, that impact on the classification of models both at the process and collaborations levels, as shown in the following.

Advances in Classifying BPMN Models. Our BPMN formalisation considers as first-class citizens BPMN specificities, reasoning on: collaboration, process and sub-process levels; asynchronous communication; and the BPMN completion notion that distinguishes the effect of end and terminate events.

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 compared to the one defined on Petri Nets and applied to the Petri Nets resulting from the translation of BPMN collaborations, e.g. via the mappings in [13] and in [23]. 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 tokens both on message and sequence edges. Such distinction is not considered in the 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 the 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 frameworks based on Petri Nets has another limitation, concerning the management of the terminate event. Most of the available mappings, such as the ones in [23] and [20], do not consider the termination end event, while in the one provided in [13] 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 that include the terminate event, and more in general on all models including unsafe sub-processes. Nevertheless, given the local nature of Petri Nets transitions, such cancellation patterns are difficult to handle. This is confirmed in [19], 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 precise classification as synthesised by the Euler diagrams in Fig. 4(a) and Fig. 4(b).

In particular, Fig. 4(a) depicts the 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
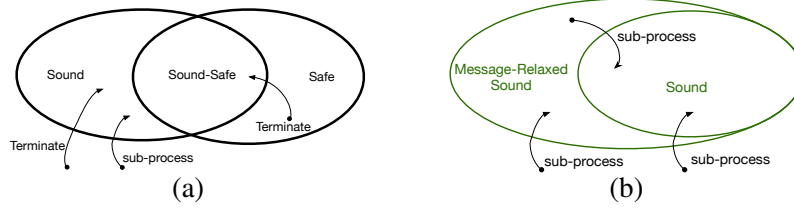
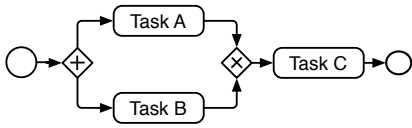Figure 4: Reasoning at process (a) and collaboration (b) level.
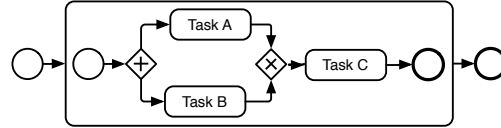

Figure 5: Unsound process.

Figure 6: Sound process with an unsound sub-process.

render a model that was unsound sound. For example, let us consider the process in Fig. 5; it is a simple process that first runs in parallel Task A and Task B, 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. 5 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. 6 shows a simple process model where the unsound process in Fig. 5 is included in the sub-process. Notably, a sub-process is not syntactic sugar. 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 [27]. Thus, it results that the model in Fig. 6 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. 4(b). Worth to notice is the possibility to transform with a small change an unsound collaboration into a sound one. In Fig. 7, 8 and 9 we report simple examples showing the impact of sub-processes. The models are rather similar, but according to our classification their behaviours are significantly different. The
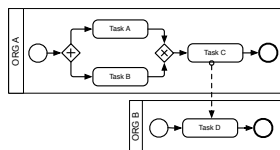
Figure 7: Unsound and message-relaxed unsound.
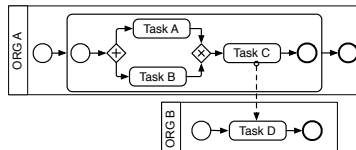
Figure 8: Unsound and message-relaxed sound.
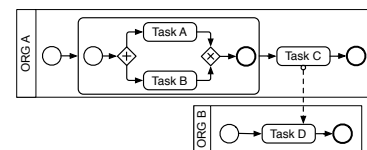
Figure 9: Sound and message-relaxed sound.

collaboration model in Fig. 7 is both unsound and message-relaxed unsound. Now, let consider another model obtained from that in Fig. 7 by introducing a sub-process. The resulting collaboration, shown in Fig. 8, is message-relaxed sound, since the use of the sub-process mitigates the problem, but it is unsound. In fact, when the two processes reach a completed configuration, there will be a pending message, since task C sends two messages and only one will be consumed by task D. Notice also that the process in

ORG A is unsafe. Thus, even if sound, the whole collaboration is unsafe. Differently, Fig. 9 shows that enclosing within a sub-process the part of the model generating multiple tokens has a positive effect on the soundness of the model. The collaboration is sound since only one message is sent and there are no pending messages. Message-relaxed soundness is also satisfied since it is implied by soundness.

4 Formal Framework

This section presents our formal semantics of BPMN collaborations. To simplify its formal definition, we rely on a textual representation of collaboration models. This permits to have more coincise and manageable semantic rules, making it easier to achieve the classification results. The direct semantics proposed in this paper is inspired by [10] [8], but its technical definition is significantly different. In particular, configuration states are here defined according to a global perspective.

Textual Representation of BPMN Collaborations. Our textual representation of the structure of BPMN collaboration models is defined by the BNF grammar in Fig. 10. In the proposed grammar, the non-terminal symbols *C* and *P* represent *Collaboration* structures and *Process* structures, 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. 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 the $\parallel$ operator, of pools of the form $\mathsf{pool}(\mathsf{p}, P)$ where: p is the name that uniquely identifies the pool, and *P* is the process included in the pool.

$$
\begin{array}{rcl}
C & ::= & \mathsf{pool}(\mathsf{p}, P) \quad | \quad C_1 \| C_2 \\
P & ::= & \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \ | \ \mathsf{end}(\mathsf{e}_i, \mathsf{e}_{cmp}) \ | \ \mathsf{startRcv}(\mathsf{e}_{enb}, \mathsf{m}, \mathsf{e}_o) \ | \ \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_{cmp}) \ | \ \mathsf{terminate}(\mathsf{e}_i) \\
& | & \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1, \mathsf{e}_{o1}), \dots, (\mathsf{m}_h, \mathsf{e}_{oh})) \ | \ \mathsf{andSplit}(\mathsf{e}_i, E_o) \ | \ \mathsf{xorSplit}(\mathsf{e}_i, E_o) \ | \ \mathsf{andJoin}(E_i, \mathsf{e}_o) \\
& | & \mathsf{xorJoin}(E_i, \mathsf{e}_o) \ | \ \mathsf{task}(\mathsf{e}_i, \mathsf{e}_o) \ | \ \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \ | \ \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \ | \ \mathsf{empty}(\mathsf{e}_i, \mathsf{e}_o) \\
& | & \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \ | \ \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m}, \mathsf{e}_o) \ | \ \mathsf{subProc}(\mathsf{e}_i, P, \mathsf{e}_o) \ | \ P_1 \| P_2
\end{array}
$$

Figure 10: BNF Syntax of BPMN Collaboration Structures.

Notationally, $\mathsf{m} \in \mathbb{M}$ denotes a *message edge*, enabling message exchanges between pairs of participants in the collaboration, while $\mathsf{e} \in \mathbb{E}$ denotes a *sequence edge*, with $E \in 2^{\mathbb{E}}$ denoting a set of sequence edges. For the convenience of the reader, we refer with e_i the edge incoming to 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* (e_{enb}) and *completing* (e_{cmp}) edges, respectively. They are needed to activate processes and sub-processes, as well as to check their completion. 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.

The correspondence between the textual representation of BPMN and the graphical one is straightforward. We exemplify it by means of a Travel Agency collaboration, used throughout the paper as a running example. In particular, the textual representation of the scenario is as follows:

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

with P_C defined as follows (process structure P_{TA} is defined in a similar way):

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

Semantics of BPMN Collaborations. The syntax presented so far permits to describe the mere structure of collaborations and processes. 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 use *collaboration configuration* and *process configuration* to indicate 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 processes 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. A process configuration, instead, 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 numbers of tokens ($\mathbb{N}$ is the set of natural numbers), while a state $\delta : \mathbb{M} \to \mathbb{N}$ is a function mapping message edges to numbers of message tokens, so that $\delta(\mathsf{m}) = \mathsf{n}$ means that there are n messages of type m sent by a participant to another one that have not been received yet. Predicate $isInit(C, \sigma, \delta)$ holds if for each process P included in C the predicate $isInit(P, \sigma)$ holds, and all message edges are unmarked in δ. Predicate $isInit(P, \sigma)$ holds when the start event of P is enabled in σ, i.e. it has a token in its enabling edge, while all other sequence edges (included the enabling edges of nested sub-processes) are unmarked in σ.

Our operational semantics is defined by means of a *Labelled Transition System*. Its transition relation on collaboration configurations formalises the execution of message marking evolution according to the process evolution. The transition $\langle C, \sigma, \delta \rangle \xrightarrow{\alpha} \langle \sigma', \delta' \rangle$ means that 'the collaboration in the configuration $\langle C, \sigma, \delta \rangle$ can do a transition labelled by α and become the collaboration configuration $\langle C, \sigma', \delta' \rangle$ in doing so'. The definition of this relation relies on an auxiliary transition relation on process configurations, for which the transition $\langle P, \sigma \rangle \xrightarrow{\alpha} \sigma'$ means that 'the process in the configuration $\langle P, \sigma \rangle$ can do a transition labelled by α and become the process configuration $\langle P, \sigma' \rangle$ in doing so'. We refer the interested reader to [9] for a full account of the definition of these relations, while we report here by way of example a few operational rules:

$$\langle \mathsf{xorSplit}(\mathsf{e}, \{\mathsf{e}'\} \cup E), \sigma \rangle \xrightarrow{\varepsilon} inc(dec(\sigma, \mathsf{e}), \mathsf{e}') \qquad \text{with } \sigma(\mathsf{e}) > 0$$

$$\langle \mathsf{andSplit}(\mathsf{e}, E), \sigma \rangle \xrightarrow{\varepsilon} inc(dec(\sigma, \mathsf{e}), E) \qquad \text{with } \sigma(\mathsf{e}) > 0$$

$$\langle \mathsf{taskRcv}(\mathsf{e}, \mathsf{m}, \mathsf{e}'), \sigma \rangle \xrightarrow{?\mathsf{m}} inc(dec(\sigma, \mathsf{e}), \mathsf{e}') \qquad \text{with } \sigma(\mathsf{e}) > 0$$

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

The first three rules concern the auxiliary transition relation on process configurations. In particular, the first one is applied when a token is available in the incoming sequence edge e of a XOR split gateway. The effect of the rule is to decrement in σ the number of tokens in the edge e (by means of function *dec*) and increment (by means of function *inc*) the tokens of the edge e' non-deterministically chosen from those outgoing from the gateway. Moreover, a label ε, corresponding to a movement of tokens internal to the process, is observed. The second rule 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 edges. The third rule acts similarly on the tokens marking the incoming and outgoing edges of a receiving task element, but it produces a label $?\mathsf{m}$ corresponding to the taking place of a receiving action. The other two rules, instead, refer to the transition relation on collaboration configurations. The first one simply states that when a process P of a pool performs an internal action τ (i.e., an action ε or a termination action *kill*), the pool performs the corresponding internal action at collaboration layer. Finally, given a pool including a process P, the last

rule can be applied only if P is willing to perform a receiving action and there is at least one message m available. Of course, one message token is consumed by the transition (by means of function *dec* applied to the state δ). We will use $\rightarrow^*$ to denote the reflexive and transitive closure of the labelled transition relations introduced above.

5   Formalisation of BPMN Collaboration Properties

We provide here a rigorous characterisation, based on the BPMN formalisation presented so far, of the key properties studied in this work, both at the level of processes and collaborations.

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 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. [21]. Such a definition was given on workflow models and it is not expressive enough for BPMN, so we redefine it including all the elements defined in our semantics (i.e. not only base elements included in workflow models, but also the event-based gateway and sub-processes). Technically, well-structuredness for processes is inductively defined on the process structure, by means of many cases (reported in [9]). Well-structuredness is then extended to collaborations, by requiring each process involved in a collaboration to be well-structured.

Definition 1 (Well-structured collaborations)**.** *Let C be a collaboration, isWS(C) is inductively defined as follows: isWS(pool(p,P)) if P is well-structured; and isWS($C_1 \parallel C_2$) if isWS(C_1) and isWS(C_2).*

Running Example. Considering the proposed running example in Fig. 2, 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.   □

Safe BPMN Collaborations. Another important condition usually required is *safeness*. Its formal definition is based on 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. We also need the following definition on the safeness of a process in a given state.

Definition 2 (Current state safe process)**.** *A process configuration $\langle P, \sigma \rangle$ is current state safe (*cs-safe*) if and only if maxMarking(P, σ) $\leq$ 1.*

The definitions of safe processes and collaborations hence require that cs-safeness is preserved along the computations.

Definition 3 (Safe processes)**.** *A process P is* safe *if and only if, given σ such that isInit(P, σ), for all σ' such that $\langle P, \sigma \rangle \rightarrow^* \sigma'$ we have that $\langle P, \sigma' \rangle$ is cs-safe.*

Definition 4 (Safe collaborations)**.** *A collaboration C is* safe *if and only if, given σ and δ such that isInit(C, σ, δ), for all σ' and δ' such that $\langle C, \sigma, \delta \rangle \rightarrow^* \langle \sigma', \delta' \rangle$ we have that maxMarking(C, σ')$\leq$1.*

Running Example. Let us consider again our running example depicted in Fig. 2. 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.   □

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 of a process in a given state. The first one is possible when all marked end events are marked exactly by a single token and all sequence edges are unmarked, while the second when no token are observed in the configuration (i.e. the case of termination due to a terminate event). So, we need the following definition of soundness of a process in a given state.

Definition 5 (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) $\forall e \in marked(\sigma, end(P)) \ . \ \sigma(e) = 1$ *and isZero*(P, σ)*;*

(ii) $\forall e \in \mathsf{edges}(P) \ . \ \sigma(e) = 0.$

In the definition above, we use the following auxiliary functions and predicates: *marked*$(\sigma, end(P))$ returns the set of edges of the end events with at least one token; *isZero*(P, σ) holds when all edges in *P*, except the completing ones, are unmarked; edges(*P*) returns the edges in *P*.

Definition 6 (Sound process)**.** *A process P is* sound *if and only if, given* σ *such that isInit*(P, σ)*, 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 7 (Sound collaboration)**.** *A collaboration C is* sound *if and only if, given* σ *and* δ *such that isInit*(C, σ, δ)*, 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* $\forall \mathsf{m} \in \mathbb{M} \ . \ \delta''(\mathsf{m}) = 0.$

Thanks to the capability of our formalisation of distinguishing sequence tokens from message tokens, we define the message-relaxed soundness property. It considers sound those collaborations in which asynchronously sent messages are not properly handled by the receiver, so that at the completion of the execution there could be pending messages.

Definition 8 (Message-relaxed sound collaboration)**.** *A collaboration C is* Message-Relaxed sound *if and only if, given* σ *and* δ *such that isInit*(C, σ, δ)*, 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.*

Running Example. 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 of the customer may not be empty. However, the collaboration satisfies the message-relaxed soundness property. □

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 in the technical report [9].

Well-structuredness vs. Safeness in BPMN. We present here some of the main results of this work concerning the correlation between well-structuredness and safeness, both at process and collaboration
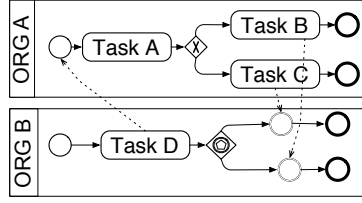
Figure 11: A safe BPMN collaboration not
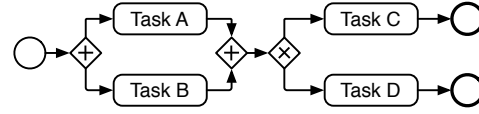well-structured.

Figure 12: An example of sound process not
well-structured.

level. Specifically, we demonstrate that all well-structured models are safe (Theorem 1 and Theorem 2), and that the vice versa does not hold. The proof that well-structured processes are safe is based on two auxiliary results: (a) a process in the initial state is cs-safe, and (b) cs-safeness is preserved by the evolution of well-structured processes.

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

Proof (sketch).   We show that if $\langle P, \sigma \rangle \to^* \sigma'$ 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 $\langle P, \sigma' \rangle$.   □

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

Proof (sketch).   We proceed by contradiction.   □

The reverse implications of Theorems 1 and 2 are not true. In fact, there are safe processes/collaborations that are not well-structured. The collaboration diagram represented in Fig. 11 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.

Well-structuredness vs. Soundness in BPMN. We demonstrate that a well-structured process is always sound (Theorem 3). The proof of this result relies on the fact that well-structured processes can always complete their execution.

Theorem 3. *Let P be a well-structured 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. 12. 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 its sequence flows. What is particularly interesting is that 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-relaxed soundness.

Theorem 4. *Let C be a collaboration, isWS(C) does not imply C is (message-relaxed) sound.*

Proof (sketch).   We proceed by contradiction.   □

Safeness vs. Soundness in BPMN. Finally, we present the relationship between safeness and soundness. Specifically we demonstrate that there are unsafe models that are sound. This is a peculiarity of BPMN,

faithfully implemented in our semantics, thanks to its capability of supporting the terminate event and (unsafe) sub-processes. Theorem 5 is demonstrated by the process in Fig. 13, while Theorem 6 by the collaboration in Fig. 14.

Theorem 5. *Let P be a process, P is unsafe does not imply P is unsound.*

Proof (sketch). We proceed by contradiction. □

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

Proof (sketch). We proceed by contradiction. □

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

7 Compositionality of Safeness and Soundness

In this section we study how the behaviour of processes affects that of the entire resulting collaboration.

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

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

Proof (sketch). We proceed by contradiction. □

The unsafeness of a collaboration, instead, 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.

Running Example. The collaboration in our example 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. □
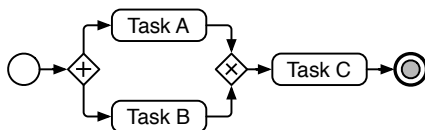

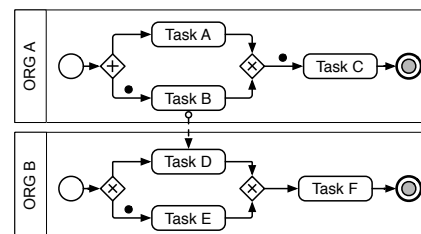Figure 13: An example of unsafe but sound process.

Figure 14: An unsafe but sound collaboration.

On Compositionality of Soundness. 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 8. *Let C be a collaboration, if some processes in C are unsound then C is unsound.*

Proof (sketch). We proceed by contradiction.   □

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.

Running Example. 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 event aborts all the running activities and removes all the tokens still present. However, the resulting collaboration is not sound, since the message lists could not be empty.   □

8   Related Work

In this paper we provide a formal characterisation of well-structuredness for BPMN models. To do that we have been inspired by the definition of well-structuredness given in [21]. Other attempts are also available in the literature [3, 16], but none of them has been extended to the collaboration level.

We also consider the safeness property. Dijkman et al. [13] discuss about safeness in Petri Nets resulting from the translation of BPMN models. In that work, safeness means that no activity will ever be enabled or running more than once concurrently. This definition is given using natural language, while in our work we give a precise characterisation of safeness for both processes and collaborations. Other approaches introducing mappings from BPMN to formal languages, such as YAWL [11], do not consider safeness.

Moreover, soundness is considered as one of the most important correctness criteria. There is a plethora of different notions of soundness in the literature, referring to different process languages and even for the same process language, e.g. for EPC a soundness definition is given by Mendling in [24], and for Workflow Nets by van der Aalst [7] provides two equivalent soundness definitions. However, these definitions cannot be directly applied to BPMN. In fact, although the BPMN process flow resembles to some extent the behaviour of Petri Nets, they are 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 a 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 that try to characterise inter-organisational soundness are available [5, 31]. However, differently from our work, these approaches do not apply to BPMN, which is the modelling notation aimed by our study. In particular, our investigation of properties at collaboration level provides novel insights with respect to the state-of-the-art of BPMN formal studies.

9   Concluding Remarks

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

References

[1] W.M.P. van der Aalst (1996): *Structural characterizations of sound workflow nets*. *Computing Science Reports* 96(23), pp. 18–22.

[2] W.M.P. van der Aalst (1997): *Verification of workflow nets*. In: *International Conference on Application and Theory of Petri Nets*, Springer, pp. 407–426, doi:10.1007/3-540-63139-9_48.

[3] W.M.P. van der Aalst (1998): *The Application of Petri Nets to Workflow Management*. *Journal of Circuits, Systems and Computers* 08(01), pp. 21–66, doi:10.1142/S0218126698000043.

[4] W.M.P. van der Aalst (1999): *Process-oriented architectures for electronic commerce and interorganizational workflow*. *Information Systems* 24(8), pp. 639–671, doi:10.1016/S0306-4379(00)00003-X.

[5] W.M.P. van der Aalst (1999): *Process-oriented architectures for electronic commerce and interorganizational workflow*. *Information Systems* 24(8), pp. 639–671, doi:10.1016/S0306-4379(00)00003-X.

[6] W.M.P. van der Aalst (2000): *Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques*. In: *Business Process Management, Models, Techniques, and Empirical Studies*, *LNCS* 1806, Springer, pp. 161–183, doi:10.1007/3-540-45594-9_11.

[7] W.M.P. van der Aalst, K. van Hee, A. ter Hofstede, N. Sidorova, H. Verbeek, M. Voorhoeve & M. Wynn (2011): *Soundness of workflow nets: classification, decidability, and analysis*. *FAC* 23(3), pp. 333–363, doi:10.1007/s00165-010-0161-4.

[8] F. Corradini, F. Fornari, A. Polini, B. Re & F. Tiezzi (2018): *A formal approach to modeling and verification of business process collaborations*. *Science of Computer Programming* 166, pp. 35–70, doi:10.1016/j.scico.2018.05.008.

[9] F. Corradini, C. Muzi, B. Re & F. Tiezzi (2018): *A Classification of BPMN Collaborations based on Safeness and Soundness Notions*. Tech.Rep., University of Camerino. Available at: `http://pros.unicam.it/documents/`.

[10] F. Corradini, A. Polini, B. Re & F. Tiezzi (2015): *An Operational Semantics of BPMN Collaboration*. In: *FACS*, *LNCS* 9539, Springer, pp. 161–180, doi:10.1007/978-3-319-28934-2_9.

[11] G. Decker, R. Dijkman, M. Dumas & L. García-Bañuelos (2008): *Transforming BPMN diagrams into YAWL nets*. In: *BPM*, *LNCS* 5240, Springer, pp. 386–389, doi:10.1007/978-3-540-85758-7_30.

[12] J. Dehnert & A. Zimmermann (2005): *On the suitability of correctness criteria for business process models*. In: *BPM*, *LNCS* 3649, Springer, pp. 386–391, doi:10.1007/11538394_28.

[13] R.M. Dijkman, M. Dumas & C. Ouyang (2008): *Semantics and analysis of business process models in BPMN*. *Information and Software Technology* 50(12), pp. 1281–1294, doi:10.1016/j.infsof.2008.02.006.

[14] M. Dumas, M. La Rosa, J. Mendling, R. Mäesalu, H.A. Reijers & N. Semenenko (2012): *Understanding business process models: the costs and benefits of structuredness*. In: *CAISE*, *LNCS* 7328, Springer, pp. 31–46, doi:10.1007/978-3-642-31095-9_3.

[15] M. Dumas, M. La Rosa, J. Mendling & H.A. Reijers (2018): *Fundamentals of Business Process Management, Second Edition*. Springer, doi:10.1007/978-3-662-56509-4.

[16] N. El-Saber (2015): *CMMI-CM compliance checking of formal BPMN models using Maude*. Ph.D. thesis, University of Leicester - Department of Computer Science.

[17] C. Favre & H. Völzer (2010): *Symbolic execution of acyclic workflow graphs*. Business Process Management, pp. 260–275, doi:10.1007/978-3-642-15618-2_19.

[18] K. van Hee, O. Oanea, A. Serebrenik, N. Sidorova & M. Voorhoeve (2006): *History-based joins: Semantics, soundness and implementation*. In: *BPM*, Springer, pp. 225–240, doi:10.1007/11841760_16.

[19] A.H.M. ter Hofstede (2002): *Workflow patterns: On the expressive power of (petri-net-based) workflow languages*. Ph.D. thesis, University of Aarhus.

[20] A. Kheldoun, K. Barkaoui & M. Ioualalen (2017): *Formal verification of complex business processes based on high-level Petri nets*. Information Sciences 385-386, pp. 39–54, doi:10.1016/j.ins.2016.12.044.

[21] B. Kiepuszewski, A.H.M. ter Hofstede & C.J. Bussler (2000): *On structured workflow modelling*. In: *Information Systems Engineering, 25 Years of CAiSE*, LNCS 9539, Springer, pp. 431–445, doi:10.1007/3-540-45140-4_29.

[22] A. Lindsay, D. Downs & K. Lunn (2003): *Business processes—attempts to find a definition*. Information and Software Technology 45(15), pp. 1015–1019, doi:10.1016/S0950-5849(03)00129-0.

[23] K. Matthias & W. Mathias (2016): *Behavioural Models - From Modelling Finite Automata to Analysing Business Processes*. Springer, doi:10.1007/978-3-319-44960-9.

[24] J. Mendling (2007): *Detection and prediction of errors in EPC business process models*. Ph.D. thesis, Wirtschaftsuniversität Wien Vienna.

[25] M. Muehlen & J. Recker (2008): *How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation*. In: *CAISE*, LNCS 5074, Springer, pp. 465–479, doi:10.1007/978-3-540-69534-9_35.

[26] T. Murata (1989): *Petri nets: Properties, analysis and applications*. IEEE Proceedings 77(4), pp. 541–580, doi:10.1109/5.24143.

[27] OMG (2011): *Business Process Model and Notation (BPMN V 2.0)*.

[28] A. Polyvyanyy, L. García-Bañuelos & M. Dumas (2012): *Structuring acyclic process models*. Information Systems 37(6), pp. 518–538, doi:10.1016/j.is.2011.10.005.

[29] A. Polyvyanyy, L. Garcia-Banuelos, D. Fahland & M. Weske (2014): *Maximal Structuring of Acyclic Process Models*. The Computer Journal 57(1), pp. 12–35, doi:10.1093/comjnl/bxs126.

[30] T. Prinz (2013): *Fast soundness verification of workflow graphs*. In: *ZEUS*, LNCS 1029, Springer, pp. 31–38.

[31] J. Roa, O. Chiotti & P. Villarreal (2011): *A verification method for collaborative business processes*. In: *International Conference on Business Process Management*, LNBIP 99, Springer, pp. 293–305, doi:10.1007/978-3-642-28108-2_29.

[32] G. Rozenberg & J. Engelfriet (1998): *Elementary net systems*. In: *Lectures on Petri Nets I: Basic Models*, Springer, pp. 12–121, doi:10.1007/3-540-65306-6_14.

[33] A. Suchenia, T. Potempa, A. Ligeza, K. Jobczyk & K. Kluza (2017): *Selected Approaches Towards Taxonomy of Business Process Anomalies*. In: *Advances in Business ICT: New Ideas from Ongoing Research*, SCI 658, Springer, pp. 65–85, doi:10.1007/978-3-319-47208-9_5.