Animating Multiple Instances in BPMN Collaborations: From Formal Semantics to Tool Support

Flavio Corradini, Chiara Muzi, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi(✉)

School of Science and Technology, University of Camerino, Camerino, Italy
{flavio.corradini,chiara.muzi,barbara.re,lorenzo.rossi, francesco.tiezzi}@unicam.it

Abstract. The increasing adoption of modelling methods contributes to a better understanding of the flow of processes, from the internal behaviour of a single organisation to a wider perspective where several organisations exchange messages. In this regard, BPMN collaboration is a suitable modelling abstraction. Even if this is a widely accepted notation, only a limited effort has been expended in formalising its semantics, especially for what it concerns the interplay among control features, data handling and exchange of messages in scenarios requiring multiple instances of interacting participants. In this paper, we face the problem of providing a formal semantics for BPMN collaborations including multiple instances, while taking into account the data perspective. Beyond defining a novel formalisation, we also provide a BPMN collaboration animator tool faithfully implementing the formal semantics. Its visualisation facilities support designers in debugging multi-instance collaboration models.

1 Introduction

Nowadays, modelling is recognised as an important practice also in supporting the continuous improvement of IT systems. In particular, IT support for collaborative systems, where participants can cooperate and share information, demands for a clear understanding of interactions and data exchanges. To ensure proper carrying out of such interactions, the participants should be provided with enough information about the messages they must or may send in a given context. This is particularly important when multiple instances of interacting participants are involved. In this regard, BPMN [1] collaboration diagrams result to be an effective way to reflect how multiple participants cooperate to reach a shared goal.

Even if widely accepted, a major drawback of BPMN is related to the complexity of the semi-formal definition of its meta-model and the possible misunderstanding of its execution semantics defined by means of natural text description, sometimes containing misleading information [2]. This becomes a more prominent issue as we consider BPMN supporting tools, such as animators, simulators

© Springer Nature Switzerland AG 2018
M. Weske et al. (Eds.): BPM 2018, LNCS 11080, pp. 83–101, 2018.
https://doi.org/10.1007/978-3-319-98648-7_6

and enactment tools, whose implementation of the execution semantics may not be compliant with the standard and be different from each other, thus undermining models portability and tools effectiveness.

To overcome these issues, several formalisations have been proposed, mainly focussing on the control flow perspective (e.g., [3–7]). Less attention has been paid to provide a formal semantics capturing the interplay between control features, message exchanges, and data. These perspectives are strongly related, especially when a participant interacts with multi-instance participants. In fact, to achieve successful collaboration interactions, it is required to deliver the messages arriving at the receiver side to the appropriate instances. As messages are used to exchange data between participants, the BPMN standard fosters the use of the content of the messages themselves to correlate them with the corresponding instances. Thus, the data perspective plays a crucial role when considering multi-instance collaborations. Despite this, no formal semantics that considers all together these key aspects of BPMN collaboration models has been yet proposed in the literature.

In this work, we aim at filling this gap by providing an operational semantics of BPMN collaboration models including multi-instance participants, while taking into account the data perspective, considering both data objects and data-based decision gateways. Moreover, we go beyond the mere formalisation, by developing an animator tool that faithfully implements the proposed formal semantics and visualises the execution of multi-instance collaborations. It is indeed well recognised that process animators play an important role in enhancing the understanding of business processes behaviour [8] and that, to this aim, the faithful correspondence with the semantics is essential [9], although it is not always supported [10]. Visualisation of model execution via an animator allows to understand the collaboration history, its current state (also in terms of data-object values) and possible future executions [11]. This is particularly useful in case of models that are not implemented yet [12]. Our tool, called MIDA, supports model designers in achieving a priori knowledge of collaborations behaviour. This can allow them to spot erroneous interactions, which can easily arise when dealing with multiple instances, and hence to prevent undesired executions.

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

– The definition of a formal semantics for BPMN collaborations considering control flow elements, multi-instance pools, data objects and data-based decision gateways. Besides being useful per se, as it provides a precise understanding of the ambiguous and loose points of the standard, a main benefit of this formalisation is that it paves the way for the development of tools supporting model analysis.
– The development of the MIDA tool for animating BPMN collaboration models. MIDA animation features result helpful both in educational contexts, for explaining the behaviour of BPMN elements, and in practical modelling activities, for debugging errors common in multi-instance collaborations.

The rest of the paper is organised as follows. Section 2 provides the motivations underlying the work, and presents our running example. Section 3 introduces the formal framework at the basis of our approach. Section 4 shows how the formal concepts have been practically realised in the MIDA tool. Section 5 compares our work with the related ones. Finally, Sect. 6 closes the paper with lessons learned and opportunities for future work.

2 The Interplay Between Multiple Instances, Messages and Data Objects in BPMN Collaborations

To precisely deal with multiple instances in BPMN collaboration models, it is necessary to take into account the data flow. Indeed, the creation of process *instances* can be triggered by the arrival of *messages*, which contain data. Within a process instance, data is stored in *data objects*, used to drive the instance execution. Values of data objects can be used to fill the content of outgoing messages, and vice versa, the content of incoming messages can be stored in data objects. We clarify below the interplay between such concepts. To this aim, we introduce a BPMN collaboration model, used as a running example throughout the paper, concerning the management of the paper reviewing process of a scientific conference (this is a revised version of the model in [13, Sect. 4.7.2] and [14]). The example concerns the management of a single paper, which is revised by three reviewers; of course, the management of all papers submitted to the conference requires to enact the collaboration for each paper.

The collaboration model in Fig. 1 combines the activities of three participants. The *Program Committee (PC) Chair* organises the reviewing activities. For the sake of simplicity, we assume that the considered conference has only one chair. A *Reviewer* performs the reviewing activity and, since more than one reviewer takes part in this, he/she is modelled as a process instance of a multi-instance pool. Finally, the *Contact Author* is the person who submitted the paper to the conference. The reviewing process is started by the PC chair, who assigns the paper to each reviewer (via a multi-instance sequential activity with loop cardinality set to 3 according to the number of involved reviewers for each paper). The paper is passed to the PC chair process by means of a data input. After all reviews are received, and combined in the *Reviews* data object, the chair starts their evaluation. According to the value of the *Evaluation* data object, the chair prepares the acceptance/rejection letter (stored in the *Letter* data object) or, if the paper requires further discussion, the decision is postponed. Discussion interactions are here abstracted and always result in an accept or reject decision. The chair then sends back a feedback to each reviewer, attaches the reviews to the notification letter, and sends the result to the contact author.

In this scenario, data support is crucial to precisely render the message exchanges between participants, especially because multiple instances of the *Reviewer* process are created. In fact, messages coming into this pool might

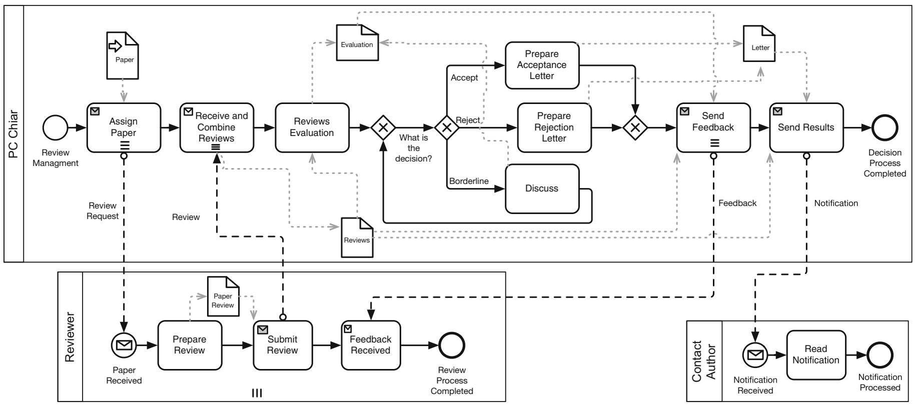


Fig. 1. Paper reviewing collaboration model.

start a new process instance, or be routed to existing instances already underway. Messages and process instances must contain enough information to determine, when a message arrives at a pool, if a new process instance is needed or, if not, which existing instance will handle it. To this aim, BPMN makes use of the concept of *correlation*: it is up to each single message to provide the information that permits to associate the message with the appropriate (possibly new) instance. This is achieved by embedding values, called *correlation data*, in the content of the message itself. Pattern-matching is used to associate a message to a distinct receiving task or event. In our example, every time the chair sends back a feedback to a reviewer, the message must contain information (in our case reviewer name and paper title) to be correlated to the correct process instance of *Reviewer*.

According to the BPMN standard, data objects do not have any direct effect on the sequence flow or message flow of processes, since tokens do not flow along data associations [1, p. 221]. However, this statement is questionable. Indeed, on the one hand, the information stored in data objects can be used to drive the execution of process instances, as they can be referred in the conditional expressions of XOR gateways to take decisions about which branch should be taken. On the other hand, data objects can be connected in input to tasks. In particular, the standard states that *"the Data Objects as inputs into the Tasks act as an additional constraint for the performance of those Tasks. The performers [...] cannot start the Task without the appropriate input"* [1, p. 183]. In both cases, a data object has an implicit indirect effect on the execution, since it can drive the decision taken by a XOR gateway or act as a guard condition on a task. For instance, in our running example, according to the value of the *Evaluation* data object, the conditional expression *What is the decision?* is evaluated and a branch of the XOR split gateway is chosen. As another example, the task *Send Results* can be executed only if an acceptance or rejection letter is stored in the *Letter* data object.

(a) Paper {title, contact, authors, body} Reviews {title, reviewers, scores, bodies}
Evaluation {title, decision} Letter {title, evaluation} PaperReview {title, score, body}

(b) ReviewRequest {title, body} Notification {title, contact, authors, evaluation, scores, bodies}
Review {reviewerName, title, score, body} Feedback {reviewerName, title, evaluation}

Fig. 2. Structures of data objects (a) and messages (b) of the paper reviewing example.

Concerning the content of data objects, the standard left underspecified its structure, in order to keep the notation independent from the kind of data structure required from time to time. We consider here a generic record structure, assuming that a data object is just a list of fields, characterised by a name and the corresponding value. Of course, a field can be used to represent the state of a data object. More complex XML-like structures, which are out of the scope of this work, can be anyway rendered resorting to nesting. The structure in terms of fields of the data objects used in our running example is specified in Fig. 2(a). Messages are structured as well; the structure of the messages specified in our example is shown in Fig. 2(b). Values can be manipulated and inserted into data object fields via assignments performed by tasks.

Guards, assignments, and structure of data objects and messages are not explicitly reported in the graphical representation of the BPMN model, but are defined as attributes of the involved BPMN elements. We provide information on their definition and functioning in Sect. 3, and show how MIDA users can specify them in Sect. 4.

3 A Formal Account of Multi-instance Collaborations

In this section we formalise the semantics of BPMN collaborations supporting multiple instances. We focus on those BPMN elements, informally presented in the previous section, that are strictly needed to deal with multiple instantiation of collaborations, namely multi-instance pools, message exchange events and tasks, and data objects; additionally, in order to define meaningful collaborations, we also consider some core BPMN elements, whose preliminary formalisation has been given in [15,16].

To simplify the formal treatment of the semantics, we resort to a textual representation of BPMN models, which is more manageable for writing operational rules than the graphical notation. Notice that we do not propose an alternative modelling notation, but we just define a Backus-Naur Form (BNF) syntax of BPMN model structures.

Textual Notation of BPMN Collaborations
We report in Fig. 3 the BNF syntax defining the textual notation of BPMN collaboration models. This syntax only describes the structure of models, without taking into account all those aspects that come into play to describe the model semantics, such as token distribution and messages. In the proposed grammar, the non-terminal symbols C, P and A represent *Collaboration Structures*, *Process*

$$\begin{array}{l}
C ::= \mathsf{pool}(\mathsf{p}, P) \quad \mid \quad \mathsf{miPool}(\mathsf{p}, P) \quad \mid \quad C_1 \parallel C_2 \\
P ::= \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o) \mid \mathsf{startRcv}(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}_o) \mid \mathsf{end}(\mathsf{e}_i) \mid \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m} : \widetilde{\mathsf{exp}}) \mid \mathsf{terminate}(\mathsf{e}_i) \\
\quad \mid \ \mathsf{andSplit}(\mathsf{e}_i, E_o) \mid \mathsf{xorSplit}(\mathsf{e}_i, G) \mid \mathsf{andJoin}(E_i, \mathsf{e}_o) \mid \mathsf{xorJoin}(E_i, \mathsf{e}_o) \\
\quad \mid \ \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1 : \tilde{\mathsf{t}}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h : \tilde{\mathsf{t}}_h, \mathsf{e}_{oh})) \\
\quad \mid \ \mathsf{task}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{e}_o) \mid \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}_o) \mid \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{m} : \widetilde{\mathsf{exp}}, \mathsf{e}_o) \\
\quad \mid \ \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}_o) \mid \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m} : \widetilde{\mathsf{exp}}, \mathsf{e}_o) \mid P_1 \parallel P_2 \\
A ::= \epsilon \ \mid \mathsf{d.f} ::= \mathsf{exp}, A
\end{array}$$

Fig. 3. BNF syntax of BPMN collaboration structures.

Structures and *Data Assignments*, respectively. The first two syntactic categories directly refer to the corresponding notions in BPMN, while the latter refers to list of assignments used to specify updating of data objects. The terminal symbols, denoted by the sans serif font, are the typical elements of a BPMN model, i.e. pools, events, tasks and gateways.

We do not provide a direct syntactic representation of *Data Objects*. The evolution of their state during the model execution is a semantic concern (described later in this section). Thus, syntactically, only the connections between data objects and the other elements are relevant. They are rendered by references to data objects within *expressions*, used to check when a task is ready to start (graphically, the task has a connection incoming from the data object), to update the values stored in a data object (graphically, the task has a connection outgoing to the data object), and to drive the decision of a XOR split gateway. A data object is structured as a list of fields; the field f of the data object named d is accessed via $\mathsf{d.f}$.

Intuitively, a BPMN collaboration model is rendered in our syntax as a collection of pools, each one specifying a process. Formally, a collaboration C is a composition, by means of the $\parallel$ operator, of pools either of the form $\mathsf{pool}(\mathsf{p}, P)$ (for single-instance pools) or $\mathsf{miPool}(\mathsf{p}, P)$ (for multi-instance pools), where p is the name that uniquely identifies the pool, and P is the enclosed process. At process level, we use $\mathsf{e} \in \mathbb{E}$ to uniquely denote a *sequence edge*, while $E \in 2^{\mathbb{E}}$ a set of edges. For the convenience of the reader, we refer with e_i to the edge incoming in an element, with e_o to the outgoing edge, and with e_{enb} to the (spurious) edge denoting the enabled status of a start event.

In the data-based setting we consider, messages may carry values. Therefore, a sending action specifies a list of expressions whose evaluation will return a tuple of values to be sent, while a receiving action specifies a template to select matching messages and possibly assign values to data object fields. Formally, a *message* is a pair $\mathsf{m} : \tilde{\mathsf{v}}$, where $\mathsf{m} \in \mathbb{M}$ is the (unique) message name (i.e., the label of the message edge), and $\tilde{\mathsf{v}}$ is a tuple of values, with $\mathsf{v} \in \mathbb{V}$ and $\tilde{\cdot}$ denoting tuples (i.e., $\tilde{\mathsf{v}}$ stands for $\langle \mathsf{v}_1, \ldots, \mathsf{v}_n \rangle$). Sending actions have as argument a pair of the form $\mathsf{m} : \widetilde{\mathsf{exp}}$. The precise syntax of *expressions* is deliberately not specified, it is just assumed that they contain, at least, values v and data object fields $\mathsf{d.f}$. Receiving actions have as argument a pair of the form $\mathsf{m} : \tilde{\mathsf{t}}$, where $\tilde{\mathsf{t}}$ denotes a

Overall paper reviewing collaboration scenario:

$\mathsf{pool}(\mathsf{p}_{pc}, P_{pc}) \parallel \mathsf{miPool}(\mathsf{p}_r, P_r) \parallel \mathsf{pool}(\mathsf{p}_{ca}, P_{ca})$

Reviewer process :

$$\begin{aligned} P_r = {} & \mathsf{startRcv}(\mathsf{ReviewRequest}:\tilde{\mathsf{t}}_2, \mathsf{e}_{15}) \parallel \mathsf{task}(\mathsf{e}_{15}, \mathsf{true}, \mathsf{A}_6, \mathsf{e}_{16}) \parallel \\ & \mathsf{taskSnd}(\mathsf{e}_{16}, \mathsf{exp}_7, \epsilon, \mathsf{Review}:\tilde{\mathsf{exp}}_8, \mathsf{e}_{17}) \parallel \\ & \mathsf{taskRcv}(\mathsf{e}_{17}, \mathsf{true}, \epsilon, \mathsf{Feedback}:\tilde{\mathsf{t}}_3, \mathsf{e}_{18}) \parallel \mathsf{end}(\mathsf{e}_{18}) \end{aligned}$$

Templates, *expressions*, *assignments* :

$$\begin{aligned} \tilde{\mathsf{t}}_2 &= \langle ?\mathsf{ReviewRequest.title}, ?\mathsf{ReviewRequest.body} \rangle \\ A_6 &= \mathsf{PaperReview.title} := \mathsf{ReviewRequest.title}, \\ & \quad\ \mathsf{PaperReview.score} := \mathsf{assignscore}(\mathsf{ReviewRequest.body}), \\ & \quad\ \mathsf{PaperReview.body} := \mathsf{writeReview}(\mathsf{ReviewRequest.body}) \\ \mathsf{exp}_7 &= \mathsf{PaperReview.score} \neq \mathsf{null\ and\ PaperReview.body} \neq \mathsf{null} \\ \tilde{\mathsf{exp}}_8 &= \langle \mathsf{myName}(), \mathsf{PaperReview.title}, \mathsf{PaperReview.score}, \mathsf{PaperReview.body} \rangle \\ \tilde{\mathsf{t}}_3 &= \langle \mathsf{myName}(), \mathsf{ReviewRequest.title}, ?\mathsf{Feedback.evaluation} \rangle \end{aligned}$$

Fig. 4. Textual representation of the running example (an excerpt).

template, that is a sequence of expressions and formal fields used as pattern to select messages received by the pool. Formal fields are data object fields identified by the ?-tag (e.g., ?d.f) and are used to bind fields to values. In order to store the received values and allow their reuse, we associate to each message in the receiving process a data object, whose name coincides with the message name. Data objects are associated to a task by means of a conditional expression, which is a guard enabling the task execution, and a list of *assignments* A, each of which assigns the value of an expression to a data field. When there is no data object as input to a task, the guard is simply true, while if there is no data object in output to a task the list of assignments is empty (ϵ).

The XOR split gateway specifies *guard conditions* in its outgoing edges, used to decide which edge to activate according to the values of data objects. This is formally rendered as a function $G : \mathbb{E} \rightarrow \mathbb{EXP}$ mapping edges to conditional expressions, where $\mathbb{EXP}$ is the set of all expressions that includes the distinguished expression default referring to the *default sequence edge* outgoing from the gateway (it is assigned to at most one edge). When convenient, we will deal with function G as a set of pairs $(\mathsf{e}, \mathsf{exp})$.

The correspondence between the syntax used here to represent multi-instance collaborations and the graphical notation of BPMN is exemplified by means of (an excerpt of) our running example in Fig. 4, while the detailed one-to-one correspondence is reported in the companion technical report [17]. In the textual notation, to support a compositional approach, each sequence (resp. message) edge in the graphical notation 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 by the unique edge name. Notably, even if our syntax would allow to write collaborations that cannot be expressed in BPMN, we only consider those terms that are derived from BPMN models.

Semantics of BPMN Collaborations

The syntax presented so far represents the mere structure of processes and collaborations. To describe their semantics, we mark sequence edges by means of tokens [1, p. 27]. In particular, we enrich the structural information with a notion of execution state, defined by the state of each process instance (given by the marking of sequence edges and the values of data object fields) and the store of the exchanged messages. We call process configurations and collaboration configurations these stateful descriptions, which produce local and global effects, respectively, on the collaboration execution.

Formally, a *process configuration* has the form $\langle P, \sigma, \alpha \rangle$, where: P is a process structure; $\sigma : \mathbb{E} \rightarrow \mathbb{N}$ is a *sequence edge state function* specifying, for each sequence edge, the current number of tokens marking it ($\mathbb{N}$ is the set of natural numbers); and $\alpha : \mathbb{F} \rightarrow \mathbb{V}$ is the *data state function* assigning values (possibly null) to data object fields ($\mathbb{F}$ is the set of data fields and $\mathbb{V}$ the set of values). We denote by σ_0 (resp. α_0) the edge (resp. data) state where all edges are unmarked (resp. all fields are set to null). The state obtained by updating in 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 $\sigma(\mathsf{e}')$. The update of data state α is similarly defined. To simplify the definition of the operational rules, we introduce some auxiliary functions to update states. Function $\mathit{inc} : \mathbb{S}_\sigma \times \mathbb{E} \longrightarrow \mathbb{S}_\sigma$ (resp. $\mathit{dec} : \mathbb{S}_\sigma \times \mathbb{E} \longrightarrow \mathbb{S}_\sigma$), where $\mathbb{S}_\sigma$ is the set of edge states, updates a state by incrementing (resp. decrementing) by one the number of tokens marking an edge in the state. They are defined as $\mathit{inc}(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto \sigma(\mathsf{e}) + 1]$ and $\mathit{dec}(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto \sigma(\mathsf{e}) - 1]$. These functions extend in a natural ways to sets E of edges. Function $\mathit{reset} : \mathbb{S}_\sigma \times \mathbb{E} \longrightarrow \mathbb{S}_\sigma$, instead, updates an edge state by setting to zero the number of tokens marking an edge in the state: $\mathit{reset}(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto 0]$. We use the *evaluation* relation $\mathit{eval} \subseteq \mathbb{EXP} \times \mathbb{S}_\alpha \times \mathbb{V}$ to evaluate an expression over a data state. This is a relation, not a function, because an expression may contain non-deterministic operators and, in such a case, its evaluation results in one of the possible values for that expression with respect to the given data state. Notation $\mathit{eval}(\mathsf{exp}, \alpha, \mathsf{v})$ states that v is one of the possible values resulting from the evaluation of the expression exp on the data state α. This relation is not explicitly defined, since the syntax of expressions is deliberately not specified; we only assume that $\mathit{eval}(\mathsf{default}, \alpha, \mathsf{v})$ implies $\mathsf{v} = \mathit{false}$ for any α. The relation extends to tuples component-wise. Finally, relation $\mathit{upd} \subseteq \mathbb{S}_\alpha \times \mathbb{A}^n \times \mathbb{S}_\alpha$, where $\mathbb{S}_\alpha$ is the set of data states and $\mathbb{A}$ is the set of assignments, is used to update data object values. Notation $\mathit{upd}(\alpha, A, \alpha')$ states that α' is one of the possible states resulting from the update of α with assignment A. The relation is inductively defined as follows: for any α, $\mathit{upd}(\alpha, \epsilon, \alpha)$; $\mathit{upd}(\alpha, \mathsf{d.f} := \mathsf{exp}, \alpha \cdot [\mathsf{d.f} \mapsto \mathsf{v}])$ with v such that $\mathit{eval}(\mathsf{exp}, \alpha, \mathsf{v})$; and $\mathit{upd}(\alpha, (A_1, A_2), \alpha'')$ with α'' such that $\mathit{upd}(\alpha', A_2, \alpha'')$ and α' such that $\mathit{upd}(\alpha, A_1, \alpha')$.

A *collaboration configuration* has the form $\langle C, \iota, \delta \rangle$, where: C is a collaboration structure; $\iota : \mathbb{P} \rightarrow 2^{\mathbb{S}_\sigma \times \mathbb{S}_\alpha}$ is the *instance state function* mapping each pool name ($\mathbb{P}$ is the set of pool names) to a multiset of instance states (ranged over by I and containing pairs of the form $\langle\sigma, \alpha\rangle$); and $\delta : \mathbb{M} \rightarrow 2^{\mathbb{V}^n}$ is a *message state function* specifying, for each message name m, a multiset of value tuples representing the messages received along the message edge labelled by m. Function δ can be updated in a way similar to σ, enabling the definition of the following auxiliary functions. Function $add : \mathbb{S}_\delta \times \mathbb{M} \times \mathbb{V}^n \rightarrow \mathbb{S}_\delta$ (resp. $rm : \mathbb{S}_\delta \times \mathbb{M} \times \mathbb{V}^n \rightarrow \mathbb{S}_\delta$), where $\mathbb{S}_\delta$ is the set of message states, allows updating a message state by adding (resp. removing) a value tuple for a given message name in the state: $add(\delta, \mathsf{m}, \tilde{\mathsf{v}}) = \delta \cdot [\mathsf{m} \mapsto \delta(\mathsf{m}) + \{\tilde{\mathsf{v}}\}]$ and $rm(\delta, \mathsf{m}, \tilde{\mathsf{v}}) = \delta \cdot [\mathsf{m} \mapsto \delta(\mathsf{m}) - \{\tilde{\mathsf{v}}\}]$, where $+$ and $-$ are the union and substraction operations on multisets. Finally, the instance state function ι can be updated in two ways: by adding a newly created instance or by modifying an existing one: $newI(\iota, \mathsf{p}, \sigma, \alpha) = \iota \cdot [\mathsf{p} \mapsto \iota(\mathsf{p}) + \{\langle\sigma, \alpha\rangle\}]$ and $updI(\iota, \mathsf{p}, I) = \iota \cdot [\mathsf{p} \mapsto I]$.

Let us go back to our running example. The scenario in its initial state is rendered as the collaboration configuration $\langle(\mathsf{pool}(\mathsf{p}_{pc}, P_{pc}) \parallel \mathsf{miPool}(\mathsf{p}_r, P_r) \parallel \mathsf{pool}(\mathsf{p}_{ca}, P_{ca})), \iota, \delta\rangle$ where: $\iota(\mathsf{p}_{pc}) = \{\langle\sigma, \alpha\rangle\}$ with $\sigma = \sigma_0 \cdot [\mathsf{e}_{enb} \mapsto 1]$ and $\alpha = \alpha_0 \cdot [\mathsf{Paper.title}, \ldots, \mathsf{Paper.body} \mapsto \mathit{title}, \ldots, \mathit{text}]$; and $\iota(\mathsf{p}_r) = \iota(\mathsf{p}_{ca}) = \varnothing$. The α function of the p_{pc} instance is initialised with the content of the *Paper* data input.

The operational semantics is defined by means of a *labelled transition system* (LTS), whose definition relies on an auxiliary LTS on the behaviour of processes. The latter is a triple $\langle\mathcal{P}, \mathcal{L}, \rightarrow\rangle$ where: $\mathcal{P}$, ranged over by $\langle P, \sigma, \alpha\rangle$, is a set of process configurations; $\mathcal{L}$, ranged over by ℓ, is a set of *labels*; and $\rightarrow \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ is a *transition relation*. We will write $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \langle P, \sigma', \alpha'\rangle$ to indicate that $(\langle P, \sigma, \alpha\rangle, \ell, \langle P, \sigma', \alpha'\rangle) \in \rightarrow$. 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. Similarly, to further improve readability, we also omit α when it is not affected by the transition. Thus, for example, a transition $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \langle P, \sigma', \alpha\rangle$ can be written as $\langle P, \sigma, \alpha\rangle \xrightarrow{\ell} \sigma'$. The labels ℓ used by the process transition relation have the following meaning. Label τ denotes an action internal to the process, while $!\mathsf{m} : \tilde{\mathsf{v}}$ and $?\mathsf{m} : \tilde{\mathsf{et}}, A$ denote sending and receiving actions, respectively. Notation $\tilde{\mathsf{et}}$ denotes an evaluated template, that is a sequence of values and formal fields. Notably, the receiving label carries information about the data assignments A to be executed, at collaboration level, after the message m is actually received. Label $new\, \mathsf{m} : \tilde{\mathsf{et}}$ denotes taking place of a receiving action that instantiates a new process instance (i.e., it corresponds to the occurrence of a start message event in a multi-instance pool). The meaning of internal actions is as follows: ϵ denotes an internal computation concerning the movement of tokens, while *kill* denotes taking place of the termination event.

An excerpt of the operational rules defining the transition relation of the processes semantics is given in Fig. 5 (we present here the rules for the BPMN elements used in our running example; we refer to [17] for a complete account). Rule *P-Start* starts the execution of a (single-instance) 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

$\langle \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_o), \sigma, \alpha\rangle \xrightarrow{\epsilon} inc(reset(\sigma, \mathsf{e}_{enb}), \mathsf{e}_o)$	$\sigma(\mathsf{e}_{enb}) > 0$	(*P-Start*)
$\langle \mathsf{end}(\mathsf{e}_i), \sigma, \alpha\rangle \xrightarrow{\epsilon} dec(\sigma, \mathsf{e}_i)$	$\sigma(\mathsf{e}_i) > 0$	(*P-End*)
$\langle \mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}_o), \sigma, \alpha\rangle \xrightarrow{new\ \mathsf{m}:\ \tilde{\mathsf{et}}} inc(\sigma, \mathsf{e}_o)$	$eval(\tilde{\mathsf{t}}, \alpha, \tilde{\mathsf{et}})$	(*P-StartRcv*)
$\langle \mathsf{xorSplit}(\mathsf{e}_i, \{(\mathsf{e}, \mathsf{exp})\} \cup G), \sigma, \alpha\rangle \xrightarrow{\epsilon} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e})$	$\sigma(\mathsf{e}_i) > 0,$ $eval(\mathsf{exp}, \alpha, true)$	(*P-XorSplit$_1$*)
$\langle \mathsf{xorSplit}(\mathsf{e}_i, \{(\mathsf{e}, \mathsf{default})\} \cup G), \sigma, \alpha\rangle \xrightarrow{\epsilon}$ $inc(dec(\sigma, \mathsf{e}_i), \mathsf{e})$	$\sigma(\mathsf{e}_i) > 0,$ $\forall (\mathsf{e}_j, \mathsf{exp}_j) \in G\ .$ $eval(\mathsf{exp}_j, \alpha, false)$	(*P-XorSplit$_2$*)
$\langle \mathsf{xorJoin}(\{\mathsf{e}\} \cup E_i, \mathsf{e}_o), \sigma, \alpha\rangle \xrightarrow{\epsilon} inc(dec(\sigma, \mathsf{e}), \mathsf{e}_o)$	$\sigma(\mathsf{e}) > 0$	(*P-XorJoin*)
$\langle \mathsf{task}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{e}_o), \sigma, \alpha\rangle \xrightarrow{\epsilon}$ $\langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), \alpha'\rangle$	$\sigma(\mathsf{e}_i) > 0,$ $eval(\mathsf{exp}, \alpha, true),$ $upd(\alpha, A, \alpha')$	(*P-Task*)
$\langle \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}_o), \sigma, \alpha\rangle \xrightarrow{?\mathsf{m}:\ \tilde{\mathsf{et}}, A}$ $inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o)$	$\sigma(\mathsf{e}_i) > 0,$ $eval(\mathsf{exp}, \alpha, true),$ $eval(\tilde{\mathsf{t}}, \alpha, \tilde{\mathsf{et}})$	(*P-TaskRcv*)
$\langle \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{exp}', A, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}_o), \sigma, \alpha\rangle \xrightarrow{!\mathsf{m}:\tilde{\mathsf{v}}}$ $\langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), \alpha'\rangle$	$\sigma(\mathsf{e}_i) > 0,$ $eval(\mathsf{exp}', \alpha, true),$ $upd(\alpha, A, \alpha'),$ $eval(\tilde{\mathsf{exp}}, \alpha, \tilde{\mathsf{v}})$	(*P-TaskSnd*)
$\dfrac{\langle P_1, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle}{\langle P_1 \parallel P_2, \sigma, \alpha\rangle \xrightarrow{\ell} \langle \sigma', \alpha'\rangle}$	$\ell \neq kill$	(*P-Int$_1$*)

Fig. 5. BPMN process semantics.

and to reset the marking of the enabling edge. Rule *P-End* instead is enabled when there is at least one token in the incoming edge of the end event, which is then simply consumed. Rule *P-StartRcv* starts the execution of a process by producing a label denoting the creation of a new instance and containing the information for consuming a received message at the collaboration layer (see rule *C-CreateMi* in Fig. 6). Rule *P-XorSplit$_1$* is applied when a token is available in the incoming edge of a XOR split gateway and a conditional expression of one of its outgoing edges is evaluated to *true*; the rule decrements the token in the incoming edge and increments the token in the selected outgoing edge. Notably, if more edges have their guards satisfied, one of them is non-deterministically chosen. Rule *P-XorSplit$_2$* is applied when all guard expressions are evaluated to *false*; in this case the default edge is marked. Rule *P-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-Task* deals with tasks, possibly equipped with data objects. It is activated only when the guard expression is satisfied and there is a token in the incoming edge, which is then moved to the outgoing edge. The rule also updates the values of the data objects connected in output to the task. Rule *P-TaskRcv* is similar, but it produces a label corresponding to the consumption of a message. In this case, however, the data updates are not executed, because they must be done only after the message is actually received; therefore, the assignment are passed by means of the label to the collaboration layer. Rule *P-TaskSnd* sends a message, updates the data object and moves the incoming token to the outgoing edge. The produced send label is used to deliver the message at the collaboration layer. We consider tasks with atomic execution; relaxation of this requirement is shown in [17]. Finally, rule *P-Int$_1$* deals with interleaving in a standard way for process elements.

Now, the labelled transition relation on collaboration configurations formalises the message exchange and the data update according to the process evolution. The LTS is a triple $\langle \mathcal{C}, \mathcal{L}_c, \rightarrow_c \rangle$ where: $\mathcal{C}$, ranged over by $\langle C, \iota, \delta \rangle$, is a set of collaboration configurations; $\mathcal{L}_c$, ranged over by l, is a set of *labels*; and $\rightarrow_c \subseteq \mathcal{C} \times \mathcal{L}_c \times \mathcal{C}$ is a *transition relation*. We apply the same readability simplifications we use for process configuration transitions. Labels l are as follows: τ is an internal action, $!\mathsf{m}:\tilde{\mathsf{v}}$ is a sending action, and $?\mathsf{m}:\tilde{\mathsf{v}}$ and *new* $\mathsf{m}:\tilde{\mathsf{v}}$ are receiving actions. Notably, at collaboration level the receiving labels just keep track of the received message. To define the collaboration semantics, an auxiliary function is needed: $match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}})$ is a partial function performing *pattern-matching* on structured data (like in [18]), thus determining if an evaluated template $\tilde{\mathsf{et}}$ matches a tuple of values $\tilde{\mathsf{v}}$. A successful matching returns a list of assignments A, updating the formal fields in the template; otherwise, the function is undefined.

The relevant operational rules defining the transition relation of the collaboration semantics are given in Fig. 6 (the full account is in [17]). Rule *C-CreateMi* deals with instance creation in the multi-instance case. An instance is created if there is a matching message; as result, the assignments for the received data are performed, and the message is consumed. The created instance is added to the multiset of existing instances of the pool. The (omitted) single-instance case is similar, except that the instance is created only if no instance exists for the considered pool ($\iota(\mathsf{p}) = \emptyset$). The next three rules allow a single pool to evolve according to the evolution of one of its process instances $\langle P, \sigma, \alpha \rangle$. In particular, if the process instance performs an internal action (rule *C-InternalMi*) or a receiving/delivery action (rules *C-ReceiveMi* or *C-DeliverMi*), the pool performs the corresponding action at collaboration layer. As for instance creation, rule *C-ReceiveMi* can be applied only if there is at least one matching message. Recall indeed that at process level the receiving labels just indicate the willingness of a process instance to consume a received message, regardless the actual presence of messages. The delivering of messages is based on the *correlation* mechanism: the correlation data are identified by the template fields that are not formal (i.e., those fields requiring specific matching values). Moreover, when a process performs a sending action, the message state function is updated in order to deliver the sent message to the receiving participant. Finally, rule *C-Int*$_1$ permits interleaving the processes execution.

It is worth noticing that the semantics has been defined according to a global perspective. Indeed, the overall state of a collaboration is collected by functions ι

and δ of its configuration. On the other hand, the global semantics of a collaboration configuration is determined, in a compositional way, by the local semantics of the involved processes, which evolve independently from each other. The use of a global perspective simplifies *(i)* the technicalities required by the formal definition of the semantics, and *(ii)* the implementation of the animation of the overall collaboration execution. The compositional definition of the semantics, anyway, would allow to easily pass to a purely local perspective, where state functions are kept separate for each process.

4 The MIDA Animation Tool

In this section, we present our BPMN animator tool MIDA (*Multiple Instances and Data Animator*) and provide details about its implementation and use. MIDA is based on the Camunda *bpmn.io* web modeller. More precisely, we have integrated our formal framework into the *bpmn.io* token simulation plug-in. The MIDA tool, as well as its source code, user guide and examples, are freely available from http://pros.unicam.it/mida/.

MIDA is a web application written in JavaScript. Its graphical interface, shown in Fig. 7, is conceived as a modelling environment. It allows users to create BPMN models using all the facilities of the Camunda modeller. In particular, data/message structures, guards and assignments can be specified by using the *Property Panel*, which permits accessing element attributes. This information is stored in appropriate elements of the standard XML representation of the BPMN model. When the animation mode is activated, by clicking the corresponding button, one or more instances of the desired processes can be fired. To do this,

$$\frac{\langle P,\sigma_0,\alpha_0\rangle \xrightarrow{new\ \mathsf{m}:\tilde{\mathsf{et}}} \langle\sigma',\alpha'\rangle \quad \tilde{\mathsf{v}}\in\delta(\mathsf{m}) \quad match(\tilde{\mathsf{et}},\tilde{\mathsf{v}})=A \quad upd(\alpha',A,\alpha'')}{\langle \mathsf{miPool}(\mathsf{p},P),\iota,\delta\rangle \xrightarrow{new\ \mathsf{m}:\tilde{\mathsf{v}}} \langle newI(\iota,\mathsf{p},\sigma',\alpha''), rm(\delta,\mathsf{m},\tilde{\mathsf{v}})\rangle} \; (C\text{-}CreateMi)$$

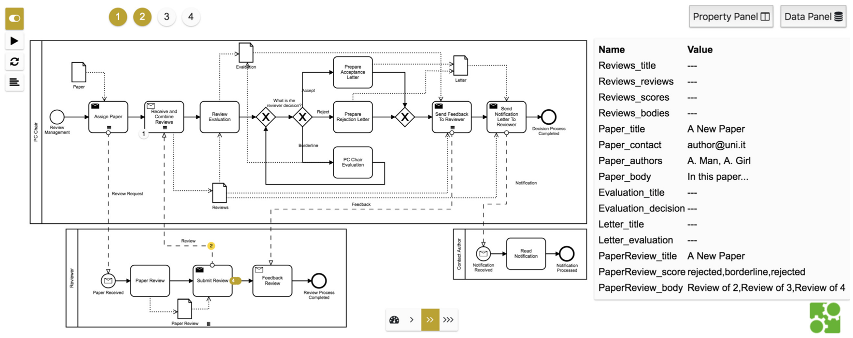
$$\frac{\iota(\mathsf{p}) = \{\langle\sigma,\alpha\rangle\}+I \qquad \langle P,\sigma,\alpha\rangle \xrightarrow{\tau} \langle\sigma',\alpha'\rangle}{\langle \mathsf{miPool}(\mathsf{p},P),\iota,\delta\rangle \xrightarrow{\tau} \langle updI(\iota,\mathsf{p},\{\langle\sigma',\alpha'\rangle\}+I),\delta\rangle} \; (C\text{-}InternalMi)$$

$$\frac{\begin{array}{c}\iota(\mathsf{p}) = \{\langle\sigma,\alpha\rangle\}+I \qquad \langle P,\sigma,\alpha\rangle \xrightarrow{?\mathsf{m}:\tilde{\mathsf{et}},A} \langle\sigma',\alpha'\rangle\\ \tilde{\mathsf{v}}\in\delta(\mathsf{m}) \qquad match(\tilde{\mathsf{et}},\tilde{\mathsf{v}})=A' \qquad upd(\alpha',(A',A),\alpha'')\end{array}}{\langle \mathsf{miPool}(\mathsf{p},P),\iota,\delta\rangle \xrightarrow{?\mathsf{m}:\tilde{\mathsf{v}}} \langle updI(\iota,\mathsf{p},\{\langle\sigma',\alpha''\rangle\}+I), rm(\delta,\mathsf{m},\tilde{\mathsf{v}})\rangle} \; (C\text{-}ReceiveMi)$$

$$\frac{\iota(\mathsf{p}) = \{\langle\sigma,\alpha\rangle\}+I \qquad \langle P,\sigma,\alpha\rangle \xrightarrow{!\mathsf{m}:\tilde{\mathsf{v}}} \langle\sigma',\alpha'\rangle}{\langle \mathsf{miPool}(\mathsf{p},P),\iota,\delta\rangle \xrightarrow{!\mathsf{m}:\tilde{\mathsf{v}}} \langle updI(\iota,\mathsf{p},\{\langle\sigma',\alpha'\rangle\}+I), add(\delta,\mathsf{m},\tilde{\mathsf{v}})\rangle} \; (C\text{-}DeliverMi)$$

$$\frac{\langle C_1,\iota,\delta\rangle \xrightarrow{l} \langle\iota',\delta'\rangle}{\langle C_1 \parallel C_2,\iota,\delta\rangle \xrightarrow{l} \langle\iota',\delta'\rangle} \; (C\text{-}Int_1)$$

Fig. 6. BPMN collaboration semantics.



Name	Value
Reviews_title	---
Reviews_reviews	---
Reviews_scores	---
Reviews_bodies	---
Paper_title	A New Paper
Paper_contact	author@uni.it
Paper_authors	A. Man, A. Girl
Paper_body	In this paper...
Evaluation_title	---
Evaluation_decision	---
Letter_title	---
Letter_evaluation	---
PaperReview_title	A New Paper
PaperReview_score	rejected,borderline,rejected
PaperReview_body	Review of 2,Review of 3,Review of 4

Fig. 7. MIDA web interface.

users have to press the *play* button depicted over each fireable start event. This creates a new token labelled with a number uniquely representing a process instance. Tokens will cross the model following the rules induced by our formal semantics. The execution of a process instance terminates once all its tokens cannot move forward. We refer to the MIDA's user guide for more details on the practical use of the tool.

MIDA animation features may be an effective support to business process designers in their modelling activities, especially when multi-instance collaborations are involved. Indeed, in this context, the choice of correlation data is an error-prone task that is a burden on the shoulders of the designers. For example, let us consider the *Reviewer* participant in our running scenario; if the template within the task for receiving the feedback would not properly specify the correlation data (e.g., $\tilde{t}_3 = \langle ?\mathsf{Feedback.reviewerName}, ?\mathsf{Feedback.title}, ?\mathsf{Feedback.evaluation}\rangle$), the feedback messages could not be properly delivered. Indeed, each *Reviewer* instance would be able to match any feedback message, regardless the reviewer name and the paper title specified in the message. Thus, the feedback messages could be mixed up. Fortunately, MIDA allows to detect, and hence solve, this correlation issue. Similarly, MIDA helps designers to detect issues concerning the exchange of messages. In fact, malformed or unexpected messages may introduce deadlocks in the execution flow, which can be easily identified by looking for blocked tokens in the animation. For instance, in the running example a feedback message without the evaluation field would be never consumed by a receiving task of the *Reviewer* instances. Finally, since our animation is based on data object values, also issues due to bad data handling can be detected using MIDA. For instance, let us suppose that the *Discuss* task in *PC Chair* would not be in a loop, but it would have its outgoing edge directly connected to the XOR join in its right hand side. After the execution of the *Discuss* task, the task *Send Feedback* would be performed, and the task *Send Results* would be activated. However, the guard of the latter task would not be satisfied, because the *Letter* data object would not be properly instantiated. This would cause a deadlock, which can be found out by using MIDA.

To sum up, the MIDA tool can support designers in debugging their multi-instance collaboration models, as it permits to check the evolution of data, messages and processes marking while executing the models step-by-step. Like in code debugging, the identification of the bug is still in charge of the human user.

5 Related Work

In this section we discuss the most relevant attempts in formalising multiple instances and data for BPMN models. We then compare MIDA with other animation tools.

On Formalising Multiple Instances and Data. Many works in the literature attempted to formalise the core features of BPMN. However, most of them (see, e.g., [3–7]) do not consider multiple instances and data, which are the focus of our work. Considering these features in BPMN collaborations, relevant works are [19–22]. Meyer et al. in [19] focus on process models where data objects are shared entities and the correlation mechanism is used to distinguish and refer data object instances. Use of data objects local to (multiple) instances, exchange of messages between participants, and correlation of messages are instead our focus. In [20], the authors describe a model-driven approach for BPMN to include the data perspective. Differently from us, they do not provide a formal semantics for BPMN multiple instances. Moreover, they do not use data in decision gateways. Moreover, Kheldoun et al. propose in [21] a formal semantics of BPMN covering features such as message-exchange, cancellation, multiple instantiation of sub-processes and exception handling, while taking into account data flow aspects. However, they do not consider multi-instance pools and do not address the correlation issue. Semantics of data objects and their use in decision gateways is instead proposed by El-Saber and Boronat in [22]. Differently from us, this formal treatment does not include collaborations and, hence, exchange of messages and multiple instances. Considering other modelling languages, YAWL [23] and high-level Petri nets [24] provide direct support for the multiple instance patterns. However, they lack support for handling data. In both cases, process instances are characterised by their identities, rather than by the values of their data, which are however necessary to correlate messages to running instances.

Regarding choreographies, relevant works are [25–27]. Gómez-López et al. [25] study the choreography problem derived from the synchronisation of multiple instances necessary for the management of data dependencies. Knuplesch et al. [26] introduces a data-aware collaboration approach including formal correctness criteria. However, they define the data perspective using data-aware interaction nets, a proprietary notation, instead of the wider accepted BPMN. Improving data-awareness and data-related capabilities for choreographies is the goal of Hahn et al. [27]. They propose a way to unify the data flow across participants with the data flow inside a participant. The scope of data objects is global to the overall choreography, while we consider data objects with scope local to participant instances, as prescribed by the BPMN standard. Apart from the specific differences mentioned above, our work differs from the others for the focus on collaboration diagrams, rather than on choreographies. This allows us to specifically deal with multiple process instantiation and messages correlation.

Finally, concerning the correlation mechanism, the BPMN standard and, hence, our work have been mainly inspired by works in the area of service-oriented computing (see the relationship between BPMN and WS-BPEL [28] in [1, Sect. 14.1.2]). In fact, when a service engages in multiple interactions, it is generally required to create an instance to concurrently serve each request, and correlate subsequent incoming messages to the created instances. Among the others, the COWS [18] formalism captures the basic aspects of SOC systems, and in particular service instantiation and message correlation à la WS-BPEL. From the formal point of view, correlation is realised by means of a pattern-matching function similar to that used in our formal semantics.

Business Process Animation. Relevant contributions about animation of business processes are proposed by Allweyer and Schweitzer [12], and by Signavio and Visual Paradigm. Differently from us, in their implementations they do not fully support the interplay between multiple instances, messages and data. Allweyer and Schweitzer propose a tool for animating BPMN models that, however, only considers processes, as it discards message exchanges, both semantically and graphically. In addition, gateway decisions are performed manually by users during the animation, instead of depending on data. The animator of the Signavio modeller allows users to step through the process element-by-element and to focus completely on the process flow. However, it discards important elements, such as message flows and data objects. Hence, Signavio animates only non-collaborative processes, without data-driven decisions, which instead are key features of our approach. Finally, Visual Paradigm provides an animator that supports also collaboration diagrams. This tool allows users to visualise the flow of messages and implements the semantics of receiving tasks and events, but it does not animate data evolution and multiple instances.

6 Concluding Remarks

This paper aims at answering the following research questions:

RQ1: What is the precise semantics of multi-instance BPMN collaborations?
RQ2: Can supporting tools assist designers to spot erroneous behaviours related to multiple instantiation and data handling in BPMN collaborations?

The answer to RQ1 is mainly given in Sect. 3, where we provide a novel operational semantics clarifying the interplay between control features, data, message exchanges and multiple instances. The answer to RQ2 is instead given in Sect. 4, where we propose MIDA, an animator tool, based on our formal semantics, that provides the visualisation of the behaviour of a collaboration by taking into account the data-based correlation of messages to process instances. We have shown, on our running example, that MIDA supports the identification of erroneous interactions, due e.g. to incorrect data handling or wrong message correlation.

We conclude the paper by discussing lessons learned, and the assumptions and limitations of our approach, also touching upon directions for future work.

Lessons Learned. The BPMN standard has the flavour of a framework rather than of a concrete language, because some aspects are not covered by it, but left to the designer [13]. For example, the standard left underspecified the internal structure of data objects: "*Data Object elements can optionally reference a DataState element [...] The definition of these states, e.g., possible values and any specific semantics are out of scope of this specification*" [1, p. 206]. This gap left by the BPMN standard must be filled in order to concretely deal with data in our formalisation, and hence in the animation of BPMN collaboration models. To this aim, we consider a generic record structure for data objects. Similarly, the expression language operating on data is left unspecified by the standard. This is not an issue for the formalisation, but the expression language has to be instantiated in the concrete implementation of the animator. In MIDA, for the sake of simplicity, we resort to the expression language of JavaScript, as this is the programming language used for implementing the tool. It conveniently allows, for example, to define expression operators that randomly select a value from a given set, which are used to define non-deterministic behaviours in our running example (see, e.g., operator assignscore() used by the *Prepare Review* task).

In addition, the lack of a formal semantics in the standard may lead to different interpretations of the tricky features of BPMN. In this work we aim at clarifying the interplay between multiple instances, messages and data objects. In particular, the standard provides an informal description of the mechanism used to correlate messages and process instances [1, p. 74], which we have formalised and implemented by following the solution adopted by the standard for executable business processes [28].

Assumptions and Limitations. Our formal semantics focusses on the communication mechanisms of collaborative systems, where multiple participants cooperate and share information. Thus, we have left out those features of BPMN whose formal treatment is orthogonal to the addressed problem, such as timed events and error handling. To keep our formalisation more manageable, multi-instance parallel tasks, sub-processes and data stores are left out too, despite they can be relevant for multi-instance collaborations. We discuss below what would be the impact of their addition to our work.

Let us first consider multi-instance tasks. The sequential instances case, as shown in the formalisation of our running example, can be simply dealt with as a macro; indeed, it corresponds to a task enclosed within a 'for' loop. The parallel case, instead, is more tricky. It is a common practice to consider it as a macro as well, which can be replaced by tasks between AND split and join gateways [3, 23], assuming to know at design time the number of instances to be generated. However, this replacement is no longer admissible when this kind of element is used within multi-instance pools [17], thus requiring a direct definition of the formal semantics of multi-instance parallel tasks.

Similar reasoning can be done for sub-processes, which again are not mere macros. In fact, in general, simply flattening a process by replacing its sub-process elements by their expanded processes results in a model with different behaviour. This because a sub-process, for example, delimits the scope of the enclosed data objects and confines the effect of termination events. Therefore, it would be necessary to explicitly deal with the resulting multi-layer perspective, which adds complexity to the formal treatment. The formalisation would become even more complex if we consider multi-instance sub-processes, which would require an extension of the correlation mechanism.

Moreover, we do not consider BPMN data stores, used to memorise shared information that will persist beyond process instance completion. Providing a formalisation for data stores would require to extend collaboration configurations with a further state function, dedicated to data stores. Moreover, the treatment of data assignments would become more intricate, as it would be necessary to distinguish updates of data objects from those of data stores, which affect different data state functions in the configuration.

Finally, values of data objects can be somehow "constrained" by assignments. Indeed, as mentioned above in the *Lessons learned* paragraph, assignment expressions can restrict the set of possible values that can be assigned to a data object field. Moreover, guard expressions of tasks or XOR split gateways can check if data object values respect given conditions. However, such constraints imposed on data object values are currently "hidden" in the expressions and, hence, in their evaluation. Assignments could be extended with an explicit definition of constraints in order to ease their specification and make more evident the effects of assignments on data values.

Future Work. We plan to continue our programme to effectively support modelling and animation of BPMN multi-instance collaborations, by overcoming the above limitations. More practically, we intend to enlarge the range of functionalities provided by MIDA, especially for what concerns the data perspective, and improve its usability. Moreover, we plan to exploit the formal semantics, and its implementation, to enable the verification of properties using, e.g., model checking techniques.

References

1. OMG: Business Process Model and Notation (BPMN V 2.0) (2011)
2. Suchenia, A., Potempa, T., Ligęza, A., Jobczyk, K., Kluza, K.: Selected approaches towards taxonomy of business process anomalies. In: Pełech-Pilichowski, T., Mach-Król, M., Olszak, C.M. (eds.) Advances in Business ICT: New Ideas from Ongoing Research. SCI, vol. 658, pp. 65–85. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-47208-9_5
3. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008)

4. Decker, G., Dijkman, R., Dumas, M., García-Bañuelos, L.: Transforming BPMN diagrams into YAWL nets. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 386–389. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_30
5. Wong, P.Y.H., Gibbons, J.: A process semantics for BPMN. In: Liu, S., Maibaum, T., Araki, K. (eds.) ICFEM 2008. LNCS, vol. 5256, pp. 355–374. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88194-0_22
6. Börger, E., Thalheim, B.: A method for verifiable and validatable business process modeling. In: Börger, E., Cisternino, A. (eds.) Advances in Software Engineering. LNCS, vol. 5316, pp. 59–115. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89762-0_3
7. Van Gorp, P., Dijkman, R.: A visual token-based formalization of BPMN 2.0 based on in-place transformations. Inf. Softw. Technol. **55**(2), 365–394 (2013)
8. Hermann, A., et al.: Collaborative business process management - a literature-based analysis of methods for supporting model understandability. In: WI, pp. 286–300 (2017)
9. Becker, J., Kugeler, M., Rosemann, M.: Process Management: A Guide for the Design of Business Processes. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-540-24798-2
10. Emens, R., Vanderfeesten, I., Reijers, H.A.: The dynamic visualization of business process models: a prototype and evaluation. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 559–570. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_45
11. Momotko, M., Nowicki, B.: Visualisation of (distributed) process execution based on extended BPMN. In: DEXA, pp. 280–284. IEEE (2003)
12. Allweyer, T., Schweitzer, S.: A tool for animating BPMN token flow. In: Mendling, J., Weidlich, M. (eds.) BPMN 2012. LNBIP, vol. 125, pp. 98–106. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33155-8_8
13. Weske, M.: Business Process Management. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73522-9
14. Corradini, F., Fornari, F., Muzi, C., Polini, A., Re, B., Tiezzi, F.: On avoiding erroneous synchronization in BPMN processes. In: Abramowicz, W. (ed.) BIS 2017. LNBIP, vol. 288, pp. 106–119. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59336-4_8
15. Corradini, F., Polini, A., Re, B., Tiezzi, F.: An operational semantics of BPMN collaboration. In: Braga, C., Ölveczky, P.C. (eds.) FACS 2015. LNCS, vol. 9539, pp. 161–180. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-28934-2_9
16. Corradini, F., Muzi, C., Re, B., Rossi, L., Tiezzi, F.: Global vs. local semantics of BPMN 2.0 Or-Join. In: Tjoa, A.M., Bellatreche, L., Biffl, S., van Leeuwen, J., Wiedermann, J. (eds.) SOFSEM 2018. LNCS, vol. 10706, pp. 321–336. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73117-9_23
17. Corradini, F., Muzi, C., Re, B., Rossi, L., Tiezzi, F.: Animating multiple instances in BPMN collaborations. Technical report, University of Camerino (2018). http://pros.unicam.it/mida/
18. Pugliese, R., Tiezzi, F.: A calculus for orchestration of web services. J. Appl. Log. **10**(1), 2–31 (2012)
19. Meyer, A., Pufahl, L., Fahland, D., Weske, M.: Modeling and enacting complex data dependencies in business processes. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 171–186. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40176-3_14

20. Meyer, A., et al.: Data perspective in process choreographies: modeling and execution. In: Techn. Ber. BPM Center Report BPM-13-29 (2013). BPMcenter.org
21. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal verification of complex business processes based on high-level Petri nets. Inf. Sci. **385–386**, 39–54 (2017)
22. El-Saber, N.A.: CMMI-CM compliance checking of formal BPMN models using Maude. Ph.D. thesis, Department of Computer Science (2015)
23. Wohed, P., et al.: Pattern-based analysis of UML activity diagrams. Beta, Research School for Operations Management and Logistics, Eindhoven (2004)
24. Van Der Aalst, W.M., Ter Hofstede, A.H.: YAWL: yet another workflow language. Inf. Syst. **30**(4), 245–275 (2005)
25. Gómez-López, M.T., Pérez-Álvarez, J.M., Varela-Vaca, A.J., Gasca, R.M.: Guiding the creation of choreographed processes with multiple instances based on data models. In: Dumas, M., Fantinato, M. (eds.) BPM 2016. LNBIP, vol. 281, pp. 239–251. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58457-7_18
26. Knuplesch, D., et al.: Data-aware interaction in distributed and collaborative workflows: modeling, semantics, correctness. In: CollaborateCom, pp. 223–232. IEEE (2012)
27. Hahn, M., Breitenbücher, U., Kopp, O., Leymann, F.: Modeling and execution of data-aware choreographies: an overview. Comput. Sci.-Res. Dev. **33**, 1–12 (2017)
28. OASIS WS BPEL TC: Web Services Business Process Execution Language Version 2.0. Technical report, OASIS, April 2007