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, Italy
name.surname@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 and messages exchange in scenarios requiring multiple instances of the 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 model designers when dealing with models involving constructs that make them intricate or may be source of errors. From a technical point of view, the tool is a web-application based on an extension of the *bpmn.io* modeller.

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 the interacting participants are involved. In this regard, BPMN [1] capability to model collaboration diagrams results to be an effective way to reflect on 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, 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 tool effectiveness.

To overcome these issues, several formalisations have been proposed, mainly focussing on the control flow perspective (e.g. [3,4]). Less attention has been paid to provide a formal semantics capturing the interplay between control features, messages exchange, and data. However, these perspectives are strongly related, especially when a participant interacts with multiple 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 multiple instances 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 provides the correct visualisation of multi-instance collaborations execution. It is indeed well recognised that process animators play an important role in enhancing the understanding of business processes behaviour [5,6] and that, to this aim, the faithful correspondence with the semantics is essential [7], although it is not always supported [8]. 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 execution [9]. This is particularly useful in case of models not executable in practice, in order to disambiguate different model interpretations by analysing the token flow [10]. More specifically, our tool, called MIDA, supports model designers in achieving a priori knowledge of collaboration behaviour that 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 beyond control flow multi-instance pools, data objects and data-based decision gateways. Besides be 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 designers in models analysis.
– The development of the MIDA software tool for animating BPMN collaboration models, based on the *bpmn.io* web modeller. MIDA animation features visualise process instances and their data, resulting helpful both in educational contexts, for explaining the behaviour of BPMN elements, and in practical modelling activities, for debugging errors common in multi-instance collaboration models.

The rest of the paper is organised as follows. Sec. 2 provides the motivations underlying our work, and illustrates the key BPMN concepts dealt with in the paper by means of a multi-instance scenario used as a running example throughout the paper. Sec. 3 introduces the formal framework at the basis of our approach. Sec. 4 shows how the formal concepts have been practically realised in our supporting tool MIDA. Sec. 5 compares our work with the related ones. Finally, Sec. 6 concludes the paper.

The rest of the paper is organised as follows. Sec. 2 provides the motivations underlying our work, and illustrates the key BPMN concepts dealt with in the paper by means of a multi-instance scenario used as a running example throughout the paper. Sec. 3 introduces the formal framework at the basis of our approach, i.e. the formalisation of the collaboration semantics including multiple instances and data. Sec. 4 shows

how the formal concepts have been practically realised in our supporting tool MIDA. Sec. 5 compares our work with the related ones. Finally, Sec. 6 concludes the paper.

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

To precisely deal with multiple instance in BPMN collaboration models, it is necessary to take into account the data flow. Indeed, the creation of process *instances* is 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 [11, Sec. 4.7.2] and [12]). 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 in Fig. 1 combines the activities of three participants. The *Program Committee (PC) Chair* organises the reviewing activities. For the sake of presentation, we assume that the considered conference has only one chair, whose behaviour is represented by the process within the PC Chair pool. A *Reviewer* is a person with knowledge in some of the conference topics. This role is modelled as a multi-instance pool. Indeed, each process instance describes the tasks that one of the three process reviewers has to accomplish to complete her/his assignment. Finally, the *Corresponding Author* is the person who submitted the paper to the conference and acts on behalf of the other authors.

The reviewing process is started by the chair, who assigns (via a multi-instance sequential activity[1]) a paper to each reviewer. The paper is passed to the chair process by means of a data input. After all reviews of the paper 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. After the review letter is prepared, the chair 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 model the message exchanges between participants, especially because multiple instances of the *Reviewer* process are created. In fact, messages coming into this pool might 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

---

[1] The assignment of the paper to reviewers could have been realised via a parallel multiple instance task. However, this feature has a tricky behaviour when used to interact with multiple instances, as we will explain in Sec. 6.
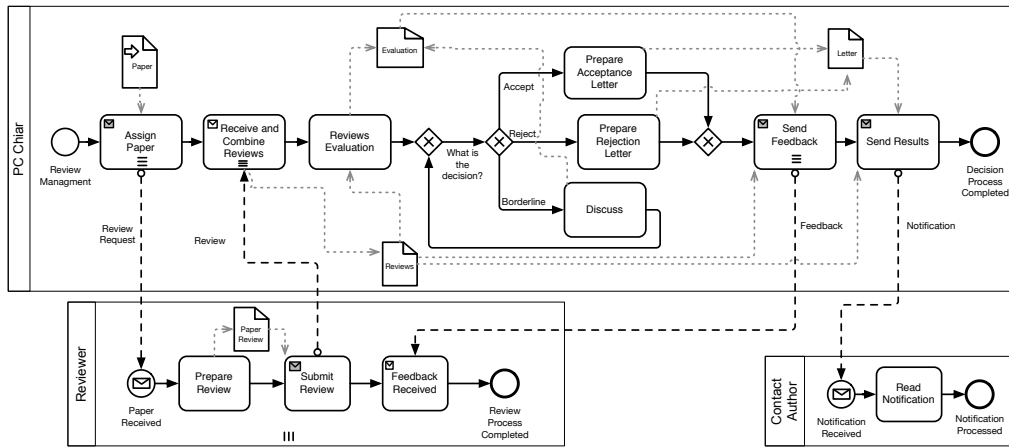
Fig. 1: Paper reviewing collaboration model.

aim, BPMN makes use of the concept of *correlation*: it is up to each single message to provide a form of context 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 (or possibly a more sophisticated mechanism) 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 the reviewer name and the paper title) to be correlated to the correct process instance of the *Reviewer* pool.

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. 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 still determine when the Tasks will be performed, but they are now constrained in that they cannot start the Task without the appropriate input"*([1], p. 183). In both cases, a data object does not directly influence the flow of the process instance, in the sense that a token cannot mark a data object, but it has an implicit, indirect effect on the execution, since it can affect 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 the *Letter* data object is in the required state, i.e. an acceptance or rejection letter is stored in the object.
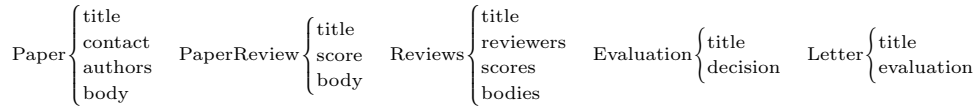
4

$$\text{Paper} \begin{cases} \text{title} \\ \text{contact} \\ \text{authors} \\ \text{body} \end{cases} \quad \text{PaperReview} \begin{cases} \text{title} \\ \text{score} \\ \text{body} \end{cases} \quad \text{Reviews} \begin{cases} \text{title} \\ \text{reviewers} \\ \text{scores} \\ \text{bodies} \end{cases} \quad \text{Evaluation} \begin{cases} \text{title} \\ \text{decision} \end{cases} \quad \text{Letter} \begin{cases} \text{title} \\ \text{evaluation} \end{cases}$$

Fig. 2: Data objects structure of the running example.

$$\text{ReviewRequest} \begin{cases} \text{title} \\ \text{body} \end{cases} \quad \text{Review} \begin{cases} \text{reviewerName} \\ \text{title} \\ \text{score} \\ \text{body} \end{cases} \quad \text{Notification} \begin{cases} \text{title} \\ \text{contact} \\ \text{authors} \\ \text{evaluation} \\ \text{scores} \\ \text{bodies} \end{cases} \quad \text{Feedback} \begin{cases} \text{reviewerName} \\ \text{title} \\ \text{evaluation} \end{cases}$$
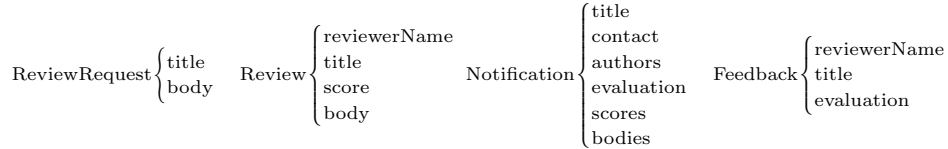
Fig. 3: Messages structure.

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 assume here a generic record structure, assuming that the data object is just a list of fields, characterised by a name and the corresponding value. More complex XML-like structures, which are out of the scope of this work, can be anyway rendered resorting to nesting. In details, the structure in terms of fields of the data objects of our running example is specified in Fig. 2. Messages are structured as well; the structure of the messages specified in our example is shown in Fig. 3.

3 A Formal Account of Multi-Instance Collaborations

In this section we formalise the semantics of BPMN collaborations supporting multiple instances. More specifically, 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 [13,14].

To simplify the formal treatment of collaborations semantics, we resort to a textual representation of BPMN models, which is more manageable for writing operational rules than the graphical notation. It is worth noticing that we do not propose an alternative modelling notation, but we just define a BNF syntax of BPMN model structures.

3.1 Textual notation of BPMN Collaborations

We report in Fig. 4 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 sent messages.

In the proposed grammar, the non-terminal symbols C, P and A represent *Collaboration Structures*, *Process Structures* and *Data Assignments*. 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,

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

Fig. 4: BNF syntax of BPMN collaboration structures.

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*. Indeed, the evolution of their state during the model execution is a semantic concern (which will be described later in Section 3.2). Thus, syntactically, only the connections between data objects and the other elements are relevant. They are rendered by the references to data objects within *expressions*, used to check when a task is ready to start (in this case, graphically, the task has a connection incoming from the data object) and to update the values stored in a data object (in this case, graphically, the task has a connection outgoing to the data object). The standard is quite loose in specifying what is the actual structure of data objects; we assume here a generic record structure, assuming that the data object is just a list of fields, characterised by a name and the corresponding value. Specifically, the field f of the data object named d is accessed via the usual notation d.f. We assume that different pools use data objects with different names (this can be easily achieved by prefixing a data object name with the name of the enclosing pool).

Intuitively, a BPMN collaboration model is rendered in our syntax as a collection of (single-instance and multi-instance) pools, and each pool contains a process. More formally, a collaboration C is a composition, by means of the $|$ 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. Notably, we have that $|E| > 1$ when E is used in joining and splitting gateways; similarly, an event-based gateway contains at least two message events, i.e. $h > 1$ in each eventBased term. For the convenience of the reader, we refer with e_i the edge incoming in an element and with e_o the edge outgoing from an element. Moreover, we refer with e_{enb} the (spurious) edge denoting the enabled status of a start event. We will use function $edges(P)$ to get the set of all edges used in the process P.

In the data-based setting we consider, messages may carry values. Therefore, a sending action specifies a list of expressions whose evaluation will return 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 $\tilde{\ }$ 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 $m : \tilde{exp}$; the precise syntax of *expressions* is deliberately not specified, it is just assumed that they contain, at least, values v and data object fields $d.f$. We consider only executable expressions, that is expressions that can be evaluated. Receiving actions have as argument a pair of the form $m : \tilde{t}$, where $\tilde{t}$ denotes a *template*, that is a sequence of expressions and formal fields used as patterns to select messages from the pool's message queue. Formal fields are data object field identified by the ?-tag (i.e., $?d.f$) and are used to bind fields to values. In order to store and allow to reuse the received values, we associate to each message in the receiving process a data object, whose name is the message name.

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 expressions which includes the distinguished expression default used to refer to the *default sequence edge* outgoing from the gateway (is assigned to at most one edge). When convenient, we will deal with function G as a set of pairs (e, exp).

Data objects are associated to a task by means of a conditional expressions, which is a guard enabling the task's 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 in input to a task, the conditional expression is simply true, while if there is no data object in output to a task the list of assignments is empty (ϵ).

The correspondence between the syntax used here to represent multi-instance collaborations and the graphical notation of BPMN is exemplified by means of our running example in Fig. 5, while the one-to-one correspondence[2] is shown in Tables 1, 2 and 3. Notably, in the textual notation there is no direct representation of the sequential multi-instance task, which is anyway simply rendered as a macro where the task is enclosed in a *for* loop (expressed by means of a pair of XOR join and split gateways, and an additional data object c_i for the loop counter). Moreover, to properly manage lists of reviewers, scores and review bodies in the PC Chair process, we use fields with vector-like data structure, equipped with the typical add() and next() functionalities. Finally, we assume an evaluate() expression to combine review scores in a decision.

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 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.

It is worth noticing that we are making the following assumptions. First, when a process is instantiated by means of a message start event, then this is the only starting event in the process. Second, processes of multi-instance pools can be instantiated only

---

[2] Notably, in the textual representation there is some information (messages content, receiving templates, data object assignments, etc.) that is not reported in the graphical notation. In fact, for the sake of understandability, according to the BPMN standard these technical details of collaborations are not part of the graphical representation, but they are part of the low-level XML representation. This information is explicitly reported in our textual representation as it is needed to properly define the execution semantics of the collaboration models.

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

$P_{pc} = \mathsf{start}(\mathsf{e}_{enb}, \mathsf{e}_1) \mid \mathsf{xorJoin}(\{\mathsf{e}_1, \mathsf{e}_1'''\}, \mathsf{e}_1') \mid$
$\qquad \mathsf{taskSnd}(\mathsf{e}_1', \mathsf{c}_1.\mathsf{c} \neq \mathsf{null}, \mathsf{c}_1.\mathsf{c} := \mathsf{c}_1.\mathsf{c} + 1, \mathsf{ReviewRequest} : \tilde{\mathsf{exp}}_1, \mathsf{e}_1'') \mid$
$\qquad \mathsf{xorSplit}(\mathsf{e}_1'', \{(\mathsf{e}_1''', \mathsf{c}_1.\mathsf{c} \leqslant 3), (\mathsf{e}_2, \mathsf{default})\}) \mid \mathsf{xorJoin}(\{\mathsf{e}_2, \mathsf{e}_2''\}, \mathsf{e}_2') \mid$
$\qquad \mathsf{taskRcv}(\mathsf{e}_2', \mathsf{c}_2.\mathsf{c} \neq \mathsf{null}, (\mathsf{c}_2.\mathsf{c} := \mathsf{c}_2.\mathsf{c} + 1, A_1), \mathsf{Review} : \tilde{\mathsf{t}}_1, \mathsf{e}_2'') \mid$
$\qquad \mathsf{xorSplit}(\mathsf{e}_2'', \{(\mathsf{e}_2''', \mathsf{c}_2.\mathsf{c} \leqslant 3), (\mathsf{e}_3, \mathsf{default})\}) \mid \mathsf{task}(\mathsf{e}_3, \mathsf{exp}_2, A_2, \mathsf{e}_4) \mid \mathsf{xorJoin}(\{\mathsf{e}_4, \mathsf{e}_{11}\}, \mathsf{e}_5) \mid$
$\qquad \mathsf{xorSplit}(\mathsf{e}_5, \{(\mathsf{e}_6, \mathsf{Evaluation.decision} = \mathsf{accept}), (\mathsf{e}_7, \mathsf{Evaluation.decision} = \mathsf{reject}),$
$\qquad\qquad\qquad (\mathsf{e}_8, \mathsf{Evaluation.decision} = \mathsf{discuss})\}) \mid$
$\qquad \mathsf{task}(\mathsf{e}_6, \mathsf{true}, A_3, \mathsf{e}_9) \mid \mathsf{task}(\mathsf{e}_7, \mathsf{true}, A_4, \mathsf{e}_{10}) \mid \mathsf{task}(\mathsf{e}_8, \mathsf{true}, A_5, \mathsf{e}_{11}) \mid \mathsf{xorJoin}(\{\mathsf{e}_9, \mathsf{e}_{10}\}, \mathsf{e}_{12}) \mid$
$\qquad \mathsf{xorJoin}(\{\mathsf{e}_{12}, \mathsf{e}_{12}'''\}, \mathsf{e}_{12}') \mid \mathsf{taskSnd}(\mathsf{e}_{12}', \mathsf{exp}_3, \mathsf{c}_3.\mathsf{c} := \mathsf{c}_3.\mathsf{c} + 1, \mathsf{Feedback} : \tilde{\mathsf{exp}}_4, \mathsf{e}_{12}'') \mid$
$\qquad \mathsf{xorSplit}(\mathsf{e}_{12}'', \{(\mathsf{e}_{12}''', \mathsf{c}_3.\mathsf{c} \leqslant 3), (\mathsf{e}_{13}, \mathsf{default})\}) \mid \mathsf{taskSnd}(\mathsf{e}_{13}, \mathsf{exp}_5, \epsilon, \mathsf{Notification} : \tilde{\mathsf{exp}}_6, \mathsf{e}_{14}) \mid$
$\qquad \mathsf{end}(\mathsf{e}_{14})$

$\tilde{\mathsf{exp}}_1 = \langle \mathsf{Paper.title}, \mathsf{Paper.body} \rangle$
$\quad A_1 = \mathsf{Reviews.title} := \mathsf{Paper.title},$
$\qquad\quad \mathsf{Reviews.reviewers} := \mathsf{add}(\mathsf{Reviews.reviewers}, \mathsf{Review.reviewerName}),$
$\qquad\quad \mathsf{Reviews.scores} := \mathsf{add}(\mathsf{Reviews.scores}, \mathsf{Review.score}),$
$\qquad\quad \mathsf{Reviews.bodies} := \mathsf{add}(\mathsf{Reviews.bodies}, \mathsf{Review.body})$
$\quad \tilde{\mathsf{t}}_1 = \langle ?\mathsf{Review.reviewerName}, \mathsf{Paper.title}, ?\mathsf{Review.score}, ?\mathsf{Review.body} \rangle$
$\mathsf{exp}_2 = \mathsf{Reviews.scores} \neq \mathsf{null} \text{ and } \mathsf{Reviews.bodies} \neq \mathsf{null} \text{ and } \mathsf{Reviews.reviewers} \neq \mathsf{null}$
$\quad A_2 = \mathsf{Evaluation.title} := \mathsf{Paper.title}, \ \mathsf{Evaluation.decision} = \mathsf{evaluate}(\mathsf{Reviews.scores})$
$\quad A_3 = \mathsf{Letter.title} := \mathsf{Paper.title}, \ \mathsf{Letter.evaluation} := \mathsf{accept}$
$\quad A_4 = \mathsf{Letter.title} := \mathsf{Paper.title}, \ \mathsf{Letter.evaluation} := \mathsf{reject}$
$\quad A_5 = \mathsf{Evaluation.title} := \mathsf{Paper.title}, \ \mathsf{Evaluation.decision} := \mathsf{evaluate}(\mathsf{Reviews.scores})$
$\mathsf{exp}_3 = \mathsf{Reviews.reviewers} \neq \mathsf{null} \text{ and } \mathsf{Evaluation.decision} \neq \mathsf{null} \text{ and } \mathsf{c}_3.\mathsf{c} \neq \mathsf{null}$
$\tilde{\mathsf{exp}}_4 = \langle \mathsf{next}(\mathsf{Reviews.reviewers}), \mathsf{Paper.title}, \mathsf{Evaluation.decision} \rangle$
$\mathsf{exp}_5 = \mathsf{Reviews.scores} \neq \mathsf{null} \text{ and } \mathsf{Reviews.bodies} \neq \mathsf{null} \text{ and } \mathsf{Letter.title} \neq \mathsf{null}$
$\qquad\quad \text{and } \mathsf{Letter.evaluation} = \mathsf{Evalution.score}$
$\tilde{\mathsf{exp}}_6 = \langle \mathsf{Paper.title}, \mathsf{Paper.contact}, \ldots, \mathsf{Reviews.bodies} \rangle$

$P_r = \mathsf{startRcv}(\mathsf{ReviewRequest} : \tilde{\mathsf{t}}_2, \mathsf{e}_{15}) \mid \mathsf{task}(\mathsf{e}_{15}, \mathsf{true}, A_6, \mathsf{e}_{16}) \mid$
$\qquad \mathsf{taskSnd}(\mathsf{e}_{16}, \mathsf{exp}_7, \epsilon, \mathsf{Review} : \tilde{\mathsf{exp}}_8, \mathsf{e}_{17}) \mid \mathsf{taskRcv}(\mathsf{e}_{17}, \mathsf{true}, \epsilon, \mathsf{Feedback} : \tilde{\mathsf{t}}_3, \mathsf{e}_{18}) \mid \mathsf{end}(\mathsf{e}_{18})$

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

$P_{ca} = \mathsf{startRcv}(\mathsf{Notification} : \tilde{\mathsf{t}}_4, \mathsf{e}_{19}) \mid \mathsf{task}(\mathsf{e}_{19}, \mathsf{true}, \epsilon, \mathsf{e}_{20}) \mid \mathsf{end}(\mathsf{e}_{20})$

$\tilde{\mathsf{t}}_4 = \langle ?\mathsf{Notification.title}, ?\mathsf{Notification.contact}, \ldots, ?\mathsf{Notification.bodies} \rangle$
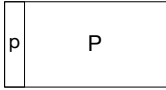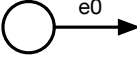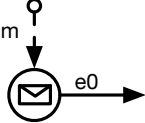
Fig. 5: Textual representation of the running example.
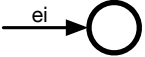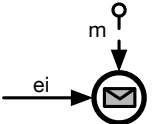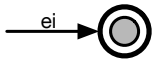
Graphical Notation	Textual Notation
	$\mathsf{pool}(\mathsf{p}, P)$
	$\mathsf{miPool}(\mathsf{p}, P)$
	$\mathsf{start}(\mathsf{e}_{enb}, \mathsf{e0})$
	$\mathsf{startRcv}(\mathsf{m}\!:\!\tilde{\mathsf{t}}, \mathsf{e0})$
	$\mathsf{end}(\mathsf{ei})$
	$\mathsf{endSnd}(\mathsf{ei}, \mathsf{m}\!:\!\tilde{\mathsf{exp}})$
	$\mathsf{terminate}(\mathsf{ei})$

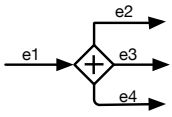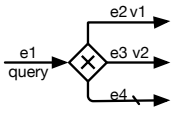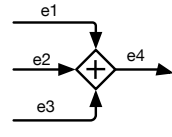Table 1: Correspondence between the graphical and textual notation of BPMN collaboration elements (pools and events).
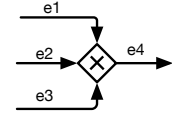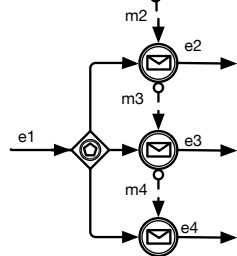
Graphical Notation	Textual Notation
	$\mathsf{andSplit}(\mathsf{e1}, \{\mathsf{e2}, \mathsf{e3}, \mathsf{e4}\})$
	$\mathsf{xorSplit}(\mathsf{e1}, \{(\mathsf{e2}, \mathsf{query} = \mathsf{v}_1), (\mathsf{e3}, \mathsf{query} = \mathsf{v}_2), (\mathsf{e4}, \mathsf{default})\})$
	$\mathsf{andJoin}(\{\mathsf{e1}, \mathsf{e2}, \mathsf{e3}\}, \mathsf{e4})$
	$\mathsf{xorJoin}(\{\mathsf{e1}, \mathsf{e2}, \mathsf{e3}\}, \mathsf{e4})$
	$\mathsf{eventBased}(\mathsf{e1}, (\mathsf{m2}\!:\!\tilde{\mathsf{t}}_2, \mathsf{e2}), (\mathsf{m3}\!:\!\tilde{\mathsf{t}}_3, \mathsf{e3}), (\mathsf{m4}\!:\!\tilde{\mathsf{t}}_4, \mathsf{e4}))$

Table 2: Correspondence between the graphical and textual notation of BPMN collaboration elements (gateways).
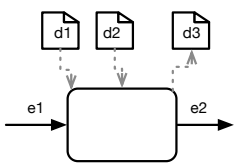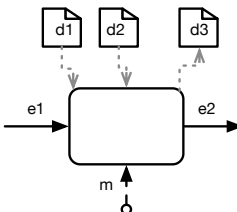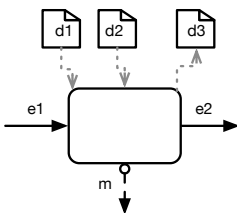
Graphical Notation	Textual Notation
	$\mathsf{task}(e1, \exp(d_1, d_2), (d_3.f_1 := \exp_1, \ldots, d_3.f_n := \exp_n), e2)$
	$\mathsf{taskRcv}(e1, \exp(d_1, d_2), (d_3.f_1 := \exp_1, \ldots, d_3.f_n := \exp_n), m : \tilde{t}, e2)$
	$\mathsf{taskSnd}(e1, \exp(d_1, d_2), (d_3.f_1 := \exp_1, \ldots, d_3.f_n := \exp_n), m : \tilde{\exp}, e2)$

Table 3: Correspondence between the graphical and textual notation of BPMN
collaboration elements (tasks).

by a message start event. The messages received by this event will carry the input data for the new instances, i.e. no data input element is used in multi-instance pools.

3.2 Semantics of BPMN Collaborations

The syntax presented so far permits to describe the mere structure of processes and collaborations. To describe their semantics we need to 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.

Formally, a *process configuration* has the form $\langle P, \sigma, \alpha \rangle$, where: P is a process structure; $\sigma : \mathbb{E} \to \mathbb{N}$ 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 $\alpha : \mathbb{F} \to \mathbb{V}$ 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 denoted by σ_0 (resp. α_0) the edge (resp. data) state where all edges are unmarked (resp. all fields are set to null); formally, $\sigma_0(\mathsf{e}) = 0 \; \forall \mathsf{e} \in \mathbb{E}$ and $\alpha_0(\mathsf{d.f}) = \mathsf{null} \;\; \forall \mathsf{d.f} \in \mathbb{F}$. The state obtained by updating in the state σ the number of tokens of the edge e to n, written as $\sigma \cdot [\mathsf{e} \mapsto \mathsf{n}]$, is defined as follows: $(\sigma \cdot [\mathsf{e} \mapsto \mathsf{n}])(\mathsf{e'})$ returns n if $\mathsf{e'} = \mathsf{e}$, otherwise it returns $\sigma(\mathsf{e'})$. The update of data state α is similarly defined. Moreover, to simplify the definition of the operational rules, we introduce some auxiliary functions to update states. Specifically, function $inc : \mathbb{S}_\sigma \times \mathbb{E} \to \mathbb{S}_\sigma$ (resp. $dec : \mathbb{S}_\sigma \times \mathbb{E} \to \mathbb{S}_\sigma$), where $\mathbb{S}_\sigma$ is the set of edge states, allows updating a state by incrementing (resp. decrementing) by one the number of tokens marking an edge in the state. Formally, they are defined as follows: $inc(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto \sigma(\mathsf{e}) + 1]$ and $dec(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto \sigma(\mathsf{e}) - 1]$. These functions extend in a natural ways to sets of edges as follows: $inc(\sigma, \varnothing) = \sigma$ and $inc(\sigma, \{\mathsf{e}\} \cup E)) = inc(inc(\sigma, \mathsf{e}), E)$; the cases for dec are similar. We also use the function $reset : \mathbb{S}_\sigma \times \mathbb{E} \to \mathbb{S}_\sigma$ that allows updating an edge state by setting to zero the number of tokens marking an edge in the state. Formally, it is defined as follows: $reset(\sigma, \mathsf{e}) = \sigma \cdot [\mathsf{e} \mapsto 0]$. Also in this case the function extends in a natural ways to sets of edges as follows: $reset(\sigma, \varnothing) = \sigma$ and $reset(\sigma, \{\mathsf{e}\} \cup E)) = reset(reset(\sigma, \mathsf{e}), E)$. We also use the *evaluation* function $[\![\mathsf{exp}]\!]_\alpha$ to evaluate an expression exp over state α: it takes an expression and a state, and returns the corresponding value. This function is not explicitly defined, since the exact syntax of expressions is deliberately not specified; we only assume that $[\![\mathsf{default}]\!]_\alpha = \mathit{false}$ for any α. The evaluation function extends to tuples component-wise. Finally, we also use the function $upd : \mathbb{S}_\alpha \times \mathbb{A}^n \to \mathbb{S}_\alpha$ to allow updating of data object values, where $\mathbb{S}_\alpha$ is the set of data states and $\mathbb{A}$ is the set of assignments. Formally, it is inductively defined as follows:

- $upd(\alpha, \epsilon) = \alpha$;
- $upd(\alpha, \mathsf{d.f} := \mathsf{exp}) = \alpha \cdot [\mathsf{d.f} \mapsto [\![\mathsf{exp}]\!]_\alpha]$;
- $upd(\alpha, A_1, A_2) = upd(upd(\alpha, A_1), A_2)$;

A *collaboration configuration* has the form $\langle C, \iota, \delta \rangle$, where: C is a collaboration structure; $\iota : \mathbb{P} \to 2^{\mathbb{S}_\sigma \times \mathbb{S}_\alpha}$ is the a *instance state function* mapping each pool name ($\mathbb{P}$

is the set of pool names) to a multiset of instance states (given as pairs of the form $\langle \sigma, \alpha \rangle$), ranged over by I; and $\delta : \mathbb{M} \to 2^{\mathbb{V}^n}$ 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 in the BPMN collaboration model. Update for δ is defined in a way similar to σ's definition, enabling the definition of the following auxiliary functions. Function $add : \mathbb{S}_\delta \times \mathbb{M} \times \mathbb{V}^n \to \mathbb{S}_\delta$ (resp. $rm : \mathbb{S}_\delta \times \mathbb{M} \times \mathbb{V}^n \to \mathbb{S}_\delta$), 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. Formally, they are defined as follows: $add(\delta, \mathsf{m}, \tilde{\mathsf{v}}) = \delta \cdot [\mathsf{m} \mapsto \delta(\mathsf{m}) + \{\tilde{\mathsf{v}}\}]$ 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 can be updated in two ways: by adding a new created instance or by modifying an existing one. Formally, we have $newI(\iota, \mathsf{p}, \sigma, \alpha) = \iota \cdot [\mathsf{p} \mapsto \iota(\mathsf{p}) + \{\langle \sigma, \alpha \rangle\}]$ and $updI(\iota, \mathsf{p}, I) = \iota \cdot [\mathsf{p} \mapsto I]$.

Our running example is then rendered as follows:

$$\langle (\mathsf{pool}(\mathsf{p}_{pc}, P_{pc}) \mid \mathsf{miPool}(\mathsf{p}_r, P_r) \mid \mathsf{pool}(\mathsf{p}_{ca}, P_{ca})), \iota, \delta \rangle$$

where:

- $\iota(\mathsf{p}_{pc}) = \{\langle \sigma, \alpha \rangle\}$ with $\sigma = \sigma_0 \cdot [\mathsf{e}_{enb} \mapsto 1]$ and $\alpha = \alpha_0 \cdot [\mathsf{Paper.title}, \ldots, \mathsf{Paper.body} \mapsto title, \ldots, text]$;
- $\iota(\mathsf{p}_r) = \varnothing$;
- $\iota(\mathsf{p}_{ca}) = \varnothing$.

The operational semantics is defined by means of a *labelled transition system* (LTS), whose definition relies on an auxiliary LTS on the behaviour of process. In our case, this is a triple $\langle \mathcal{P}, \mathcal{L}, \to \rangle$ 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* (of transitions that process configurations can perform); and $\to \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ is a *transition relation*. We will write $\langle P, \sigma, \alpha \rangle \xrightarrow{\ell} \langle P, \sigma', \alpha' \rangle$ to indicate that $(\langle P, \sigma, \alpha \rangle, \ell, \langle P, \sigma', \alpha' \rangle) \in \to$ and say that 'the process in the configuration $\langle P, \sigma, \alpha \rangle$ can do a transition labelled by ℓ and become the process configuration $\langle P, \sigma', \alpha' \rangle$ in doing so'. 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 $\langle P, \sigma, \alpha \rangle \xrightarrow{\ell} \langle P, \sigma', \alpha \rangle$ is written as $\langle P, \sigma, \alpha \rangle \xrightarrow{\ell} \sigma'$.

The labels used by the process transition relation are generated by the following production rules:

$$\ell ::= \tau \quad \mid \quad !\mathsf{m}{:}\tilde{\mathsf{v}} \quad \mid \quad ?\mathsf{m}{:}\tilde{\mathsf{et}}, A \quad \mid \quad new\ \mathsf{m}{:}\tilde{\mathsf{et}} \qquad\qquad \tau ::= \epsilon \mid\ kill$$

The meaning of labels is as follows. 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.

The operational rules defining the transition relation of the processes semantics are given in Fig. 6. We now briefly comment on some of rules. 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 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-$Terminate$ is similar, but it produces a kill label used to force the termination of the process instance. 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 rules C-$Create$ and C-$CreateMi$ in Fig. 7); as usual the effect of the rule is to increment the number of tokens in the edge outgoing from the event. Rule P-$EndSnd$ is enabled when there is at least a token in the incoming edge of the end event, which is then removed. Moreover, a send label is produced in order to deliver the produced message at the collaboration layer (see rules C-$Deliver$ and C-$DeliverMi$ in Fig. 7). Rule P-$AndSplit$ is applied when there is at least one token in the incoming edge of an AND split gateway; as result of its application the rule decrements the number of tokens in the incoming edge and increments that in each outgoing edge. 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 a token is available in the incoming edge of a XOR split gateway but all the guard expressions are evaluated to $false$; the rule decrements the token in the incoming edge and increments the token in the default outgoing edge. Rule P-$AndJoin$ decrements the tokens in each incoming edge and increments the number of tokens of the outgoing edge, when each incoming edge has at least one token. Rule P-$XorJoin$ is activated every time there is a token in one of the incoming edges, which is then moved to the outgoing edge. Rule P-$EventG$ is activated when there is a token in the incoming edge and there is a message m_j to be consumed, so that the application of the rule moves the token from the incoming edge to the outgoing edge corresponding to the received message. A label corresponding to the consumption of a message is observed. Rule P-$Task$ deals with atomic tasks, possibly equipped with data objects. It is activated only when there is a token in the incoming edge, which is then moved to the outgoing edge, and the guard expression is satisfied. The rule also updates the values of the data objects connected in output to the task. Similarly, rule P-$TaskRcv$ is applied only when there is a token in the incoming edge and the guard expression is satisfied. Moreover, a label corresponding to the consumption of a message is observed. 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 (see rules C-$Receive$ and C-$ReceiveMi$ in Fig. 7). Rule P-$TaskSnd$ sends a message, updates the data objected and moves the incoming

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

$$\langle \mathsf{end}(\mathsf{e}_i), \sigma, \alpha \rangle \xrightarrow{\epsilon} dec(\sigma, \mathsf{e}_i) \quad \sigma(\mathsf{e}_i) > 0 \qquad (\textit{P-End})$$

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

$$\langle \mathsf{startRcv}(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}_o), \sigma, \alpha \rangle \xrightarrow{new\, \mathsf{m}\, :\, [\![\tilde{\mathsf{t}}]\!]_\alpha} inc(\sigma, \mathsf{e}_o) \qquad (\textit{P-StartRcv})$$

$$\langle \mathsf{endSnd}(\mathsf{e}_i, \mathsf{m} : \tilde{\mathsf{exp}}), \sigma, \alpha \rangle \xrightarrow{!\mathsf{m}\, :\, [\![\tilde{\mathsf{exp}}]\!]_\alpha} dec(\sigma, \mathsf{e}_i) \quad \sigma(\mathsf{e}_i) > 0 \qquad (\textit{P-EndSnd})$$

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

$$\langle \mathsf{xorSplit}(\mathsf{e}_i, \{(\mathsf{e}, \mathsf{exp})\} \cup G), \sigma, \alpha \rangle \xrightarrow{\epsilon} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}) \quad \sigma(\mathsf{e}_i) > 0, \ [\![\mathsf{exp}]\!]_\alpha = true \qquad (\textit{P-XorSplit}_1)$$

$$\langle \mathsf{xorSplit}(\mathsf{e}_i, \{(\mathsf{e}, \mathsf{default})\} \cup G), \sigma, \alpha \rangle \xrightarrow{\epsilon} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}) \quad \begin{array}{l} \sigma(\mathsf{e}_i) > 0, \\ \forall (\mathsf{e}_j, \mathsf{exp}_j) \in G \ . \ [\![\mathsf{exp}_j]\!]_\alpha = false \end{array} \qquad (\textit{P-XorSplit}_2)$$

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

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

$$\langle \mathsf{eventBased}(\mathsf{e}_i, (\mathsf{m}_1 : \tilde{\mathsf{t}}_1, \mathsf{e}_{o1}), \ldots, (\mathsf{m}_h : \tilde{\mathsf{t}}_h, \mathsf{e}_{oh})), \sigma, \alpha \rangle \xrightarrow{?\mathsf{m}_j\, :\, [\![\tilde{\mathsf{t}}_j]\!]_\alpha, \epsilon} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_{oj}) \quad \sigma(\mathsf{e}_i) > 0, 1 \leqslant j \leqslant h \qquad (\textit{P-EventG})$$

$$\langle \mathsf{task}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{e}_o), \sigma, \alpha \rangle \xrightarrow{\epsilon} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), upd(\alpha, A) \rangle \quad \sigma(\mathsf{e}_i) > 0, \ [\![\mathsf{exp}]\!]_\alpha = true \qquad (\textit{P-Task})$$

$$\langle \mathsf{taskRcv}(\mathsf{e}_i, \mathsf{exp}, A, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}_o), \sigma, \alpha \rangle \xrightarrow{?\mathsf{m}\, :\, [\![\tilde{\mathsf{t}}]\!]_\alpha, A} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o) \quad \sigma(\mathsf{e}_i) > 0, \ [\![\mathsf{exp}]\!]_\alpha = true \qquad (\textit{P-TaskRcv})$$

$$\langle \mathsf{taskSnd}(\mathsf{e}_i, \mathsf{exp}', A, \mathsf{m} : \tilde{\mathsf{exp}}, \mathsf{e}_o), \sigma, \alpha \rangle \xrightarrow{!\mathsf{m}\, :\, [\![\tilde{\mathsf{exp}}]\!]_\alpha} \langle inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o), upd(\alpha, A) \rangle \quad \sigma(\mathsf{e}_i) > 0, \ [\![\mathsf{exp}']\!]_\alpha = true \qquad (\textit{P-TaskSnd})$$

$$\langle \mathsf{interRcv}(\mathsf{e}_i, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}_o), \sigma, \alpha \rangle \xrightarrow{?\mathsf{m}\, :\, [\![\tilde{\mathsf{t}}]\!]_\alpha, \epsilon} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o) \quad \sigma(\mathsf{e}_i) > 0 \qquad (\textit{P-InterRcv})$$

$$\langle \mathsf{interSnd}(\mathsf{e}_i, \mathsf{m} : \tilde{\mathsf{exp}}, \mathsf{e}_o), \sigma, \alpha \rangle \xrightarrow{!\mathsf{m}\, :\, [\![\tilde{\mathsf{exp}}]\!]_\alpha} inc(dec(\sigma, \mathsf{e}_i), \mathsf{e}_o) \quad \sigma(\mathsf{e}_i) > 0 \qquad (\textit{P-InterSnd})$$

$$\frac{\langle P_1, \sigma, \alpha \rangle \xrightarrow{kill} \langle \sigma', \alpha' \rangle}{\langle P_1 \mid P_2, \sigma, \alpha \rangle \xrightarrow{kill} \langle reset(\sigma', edges(P_1 \mid P_2)), \alpha' \rangle} \ (\textit{P-Kill}_1) \qquad \frac{\langle P_2, \sigma, \alpha \rangle \xrightarrow{kill} \langle \sigma', \alpha' \rangle}{\langle P_1 \mid P_2, \sigma, \alpha \rangle \xrightarrow{kill} \langle reset(\sigma', edges(P_1 \mid P_2)), \alpha' \rangle} \ (\textit{P-Kill}_2)$$

$$\frac{\langle P_1, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma', \alpha' \rangle \quad \ell \neq kill}{\langle P_1 \mid P_2, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma', \alpha' \rangle} \ (\textit{P-Int}_1) \qquad \frac{\langle P_2, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma', \alpha' \rangle \quad \ell \neq kill}{\langle P_1 \mid P_2, \sigma, \alpha \rangle \xrightarrow{\ell} \langle \sigma', \alpha' \rangle} \ (\textit{P-Int}_2)$$

Fig. 6: BPMN process semantics.

token to the outgoing edge. Rule P-$InterRcv$ (resp. P-$InterSnd$) follows a behaviour similar to that of rule P-$TaskRcv$ (resp. P-$TaskSnd$). Rules P-$Kill_1$ and P-$Kill_2$ deal with the propagation of killing action on the scope of the process instance, thus resetting the marking of the instance edges. Finally, Rules P-Int_1 and P-Int_2 deal 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. In the case of collaborations, the LTS is a triple $\langle \mathcal{C}, \mathcal{L}_c, \rightarrow \rangle_c$ 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* (of transitions that collaboration configurations can perform); and $\rightarrow_c \subseteq \mathcal{C} \times \mathcal{L}_c \times \mathcal{C}$ is a *transition relation*. We will apply the same readability simplifications as we have done for process configuration transitions. The labels used by the collaboration transition relation are generated by the following production rules:

$$l ::= \tau \quad | \quad !\mathsf{m}{:}\tilde{\mathsf{v}} \quad | \quad ?\mathsf{m}{:}\tilde{\mathsf{v}} \quad | \quad new\ \mathsf{m}{:}\tilde{\mathsf{v}}$$

Notably, at collaboration level the receiving label just keeps track of the received message. To define the collaboration semantics, an additional auxiliary function is needed: $match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}})$ is a partial function performing *pattern-matching* on structured data, 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 rules defining the $match$ function are as follows:

- $match(\mathsf{v}, \mathsf{v}) = \epsilon$;
- $match(?\mathsf{d}.\mathsf{f}, \mathsf{v}) = (\mathsf{d}.\mathsf{f} ::= \mathsf{v})$
- $match((\mathsf{et}', \tilde{\mathsf{et}}), (\mathsf{v}', \tilde{\mathsf{v}})) = match(\mathsf{et}', \mathsf{v}'), match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}})$

The meaning of the rules is straightforward: an evaluated template matches against a value tuple if both have the same number of fields and corresponding fields do match; two values match only if they are identical, while a formal field matches any value.

The operational rules defining the transition relation of the collaboration semantics are given in Fig. 7. We now briefly comment on some of rules. The first two rules deal with instance creation. In the single instance case (rule *C-Create*), an instance is created only if no instance exists for the considered pool, and there is a matching message. As result, the assignments for the received data are performed, and the message is consumed. In the multi-instance case (rule *C-CreateMi*), the created instance is simply added to the multiset of existing instances of the pool. The next six rules allow a single pool, representing organisation p, 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 (rules *C-Internal* or *C-InternalMi*) or a receiving/delivery action (rules *C-Receive*, *C-ReceiveMi*, *C-Deliver* or *C-DeliverMi*), the pool performs the corresponding action at collaboration layer. As for instance creation, rules *C-Receive* and *C-ReceiveMi* can be applied only if there is at least one message action. 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. Notably, 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

16

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

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

$$\frac{\iota(\mathsf{p}) = \{\langle \sigma, \alpha \rangle\} \quad \langle P, \sigma, \alpha \rangle \xrightarrow{\tau} \langle \sigma', \alpha' \rangle}{\langle \mathsf{pool}(\mathsf{p}, P), \iota, \delta \rangle \xrightarrow{\tau} \langle updI(\iota, \mathsf{p}, \{\langle \sigma', \alpha' \rangle\}), \delta \rangle} \; (C\text{-}Internal)$$

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

$$\frac{\iota(\mathsf{p}) = \{\langle \sigma, \alpha \rangle\} \quad \langle P, \sigma, \alpha \rangle \xrightarrow{?\mathsf{m}:\tilde{\mathsf{et}},A} \langle \sigma', \alpha' \rangle \quad \tilde{\mathsf{v}} \in \delta(\mathsf{m}) \quad match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) = A'}{\langle \mathsf{pool}(\mathsf{p}, P), \iota, \delta \rangle \xrightarrow{?\mathsf{m}:\tilde{\mathsf{v}}} \langle updI(\iota, \mathsf{p}, \{\langle \sigma', upd(\alpha', (A', A)) \rangle\}), rm(\delta, \mathsf{m}, \tilde{\mathsf{v}}) \rangle} \; (C\text{-}Receive)$$

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

$$\frac{\iota(\mathsf{p}) = \{\langle \sigma, \alpha \rangle\} \quad \langle P, \sigma, \alpha \rangle \xrightarrow{!\mathsf{m}:\tilde{\mathsf{v}}} \langle \sigma', \alpha' \rangle}{\langle \mathsf{pool}(\mathsf{p}, P), \iota, \delta \rangle \xrightarrow{!\mathsf{m}:\tilde{\mathsf{v}}} \langle updI(\iota, \mathsf{p}, \{\langle \sigma', \alpha' \rangle\}), add(\delta, \mathsf{m}, \tilde{\mathsf{v}}) \rangle} \; (C\text{-}Deliver)$$

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

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

Fig. 7: BPMN collaboration semantics.

updated in order to deliver the sent message to the receiving organization. Rules *C-Int*$_1$ and *C-Int*$_2$ permit to interleave the execution of actions performed by pools of the same collaboration, so that if a part of a larger collaboration evolves, the whole collaboration evolves accordingly.

3.3 Non-atomic tasks

So far, we have only considered tasks with atomic execution. Indeed, for a given task, the evaluation of its enabling guard, the execution of its activities, the possible sending/receiving of a message, and the data object assignments, are performed atomically.

This semantics fits well in many scenarios, e.g. when a task acts on a data object representing a paper document managed by a human actor, which cannot be accessed concurrently by other actors involved in the collaboration. However, there are also some situations where non-atomic access is more suitable, e.g. when data objects are shared digital documents.

Actually, the BPMN standard is intentionally loose on this point, in order to allow the use of the modelling language in different contexts of use. To more effectively support designers, we believe that both modality of access to data objects should be included in our formalisation. This enables the identification of concurrency issues in those situations where they can arise and, at same time, it allows to not take into account such issues when in the reality they cannot occur. We show below how the atomic execution constraint is relaxed.

Form the syntactic point of view, we have to extend the syntax of processes with specific constructs representing the tasks with non-atomic access to data objects:

$$P ::= \ldots \mid \mathsf{task\_na}(\mathsf{t}, \mathsf{e}_i, \mathsf{exp}, A, \mathsf{e}_o)$$
$$\mid \mathsf{taskRcv\_na}(\mathsf{t}, \mathsf{e}_i, \mathsf{exp}, A, \mathsf{m}:\tilde{\mathsf{t}}, \mathsf{e}_o) \mid \mathsf{taskSnd\_na}(\mathsf{t}, \mathsf{e}_i, \mathsf{exp}, A, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{e}_o)$$

From the practical point of view, we can think of these as BPMN task elements with an appropriate attribute set to specify that their execution is non atomic. Notably, now each task specifies a name (range over by t).

Now, to achieve a non-atomic semantics for the above elements we have only to include in process and collaboration configurations information about the status of tasks, which can be *idle* (i), *running* (r) or exchanged message (m). Formally, this is rendered as an additional state function, denoted by ψ, mapping task names to their execution status. As usual, to change the status of a task, we use dedicated functions defined as follows:

- $setIdle(\psi, \mathsf{t}) = \psi \cdot [\mathsf{t} \mapsto i]$
- $setRun(\psi, \mathsf{t}) = \psi \cdot [\mathsf{t} \mapsto r]$
- $setMsg(\psi, \mathsf{t}) = \psi \cdot [\mathsf{t} \mapsto m]$

Rules for non-atomic tasks are reported in Fig. 8. Basically, the rule for non-communicating tasks is split in two rules: $P\text{-}RunTask$ dealing with task activation and $P\text{-}CmpTask$ dealing with task completion. Notice that the data update assignments are performed at the completion time. Similarly, the rules for receiving/sending tasks are split in three rules: one for task activation, one for receiving/sending the message while the task is running, and one for task completion (and, hence, data updating).

No change are required at the collaboration layer, apart for the addition of ψ in the collaboration configurations, which anyway is not actively involved in collaboration transitions.

4 Support tool for animation

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

$$\langle \mathsf{task\_na}(\mathsf{t}, \mathsf{e}_i, \exp, A, \mathsf{e}_o), \sigma, \alpha, \psi \rangle \xrightarrow{\epsilon} \langle dec(\sigma, \mathsf{e}_i), setRun(\psi, \mathsf{t}) \rangle \quad \begin{array}{c} \sigma(\mathsf{e}_i) > 0, \psi(\mathsf{t}) = i \\ [\![\exp]\!]_\alpha = true \end{array} \quad (P\text{-}RunTask)$$

$$\langle \mathsf{task\_na}(\mathsf{t}, \mathsf{e}_i, \exp, A, \mathsf{e}_o), \sigma, \alpha, \psi \rangle \xrightarrow{\epsilon} \langle inc(\sigma, \mathsf{e}_o), upd(\alpha, A), setIdle(\psi, \mathsf{t}) \rangle \quad \qquad (P\text{-}CmpTask)$$

$$\langle \mathsf{taskRcv\_na}(\mathsf{t}, \mathsf{e}_i, \exp, A, \mathsf{m}{:}\tilde{\mathsf{t}}, \mathsf{e}_o), \sigma, \alpha, \psi \rangle \xrightarrow{\epsilon} \langle dec(\sigma, \mathsf{e}_i), setRun(\psi, \mathsf{t}) \rangle \quad \begin{array}{c} \sigma(\mathsf{e}_i) > 0, \psi(\mathsf{t}) = i \\ [\![\exp]\!]_\alpha = true \end{array} \quad (P\text{-}RunTaskRcv)$$

$$\langle \mathsf{taskRcv\_na}(\mathsf{t}, \mathsf{e}_i, \exp, A, \mathsf{m}{:}\tilde{\mathsf{t}}, \mathsf{e}_o), \sigma, \alpha, \psi \rangle \xrightarrow{?\mathsf{m}{:}[\![\tilde{\mathsf{t}}]\!]_\alpha, \epsilon} \langle inc(\sigma, \mathsf{e}_o), setMsg(\psi, \mathsf{t}) \rangle \quad \psi(\mathsf{t}) = r \quad (P\text{-}TaskRcv)$$

$$\langle \mathsf{taskRcv\_na}(\mathsf{t}, \mathsf{e}_i, \exp, A, \mathsf{m}{:}\tilde{\mathsf{t}}, \mathsf{e}_o), \sigma, \alpha, \psi \rangle \xrightarrow{\epsilon} \langle inc(\sigma, \mathsf{e}_o), upd(\alpha, A), setIdle(\psi, \mathsf{t}) \rangle \quad \psi(\mathsf{t}) = m \quad (P\text{-}CmpTaskRcv)$$

$$\langle \mathsf{taskSnd\_na}(\mathsf{t}, \mathsf{e}_i, \exp', A, \mathsf{m}{:}\tilde{\exp}, \mathsf{e}_o), \sigma, \alpha, \psi \rangle \xrightarrow{\epsilon} \langle dec(\sigma, \mathsf{e}_i), setRun(\psi, \mathsf{t}) \rangle \quad \begin{array}{c} \sigma(\mathsf{e}_i) > 0, \psi(\mathsf{t}) = i \\ [\![\exp']\!]_\alpha = true \end{array} \quad (P\text{-}RunTaskSnd)$$

$$\langle \mathsf{taskSnd\_na}(\mathsf{t}, \mathsf{e}_i, \exp', A, \mathsf{m}{:}\tilde{\exp}, \mathsf{e}_o), \sigma, \alpha, \psi \rangle \xrightarrow{!\mathsf{m}{:}[\![\tilde{\exp}]\!]_\alpha} \langle inc(\sigma, \mathsf{e}_o), setMsg(\psi, \mathsf{t}) \rangle \quad \psi(\mathsf{t}) = r \quad (P\text{-}TaskSnd)$$

$$\langle \mathsf{taskSnd\_na}(\mathsf{t}, \mathsf{e}_i, \exp', A, \mathsf{m}{:}\tilde{\exp}, \mathsf{e}_o), \sigma, \alpha, \psi \rangle \xrightarrow{\epsilon} \langle inc(\sigma, \mathsf{e}_o), upd(\alpha, A), setIdle(\psi, \mathsf{t}) \rangle \quad \psi(\mathsf{t}) = m \quad (P\text{-}CmpTaskSnd)$$

Fig. 8: BPMN semantics of non-atomic tasks.

on the Camunda *bpmn.io* web modeller. More precisely, we have integrated our formal framework into the *bpmn.io* token simulation plug-in[3]. We have enriched this plug-in with a wider set of BPMN elements and redefined their semantics. Moreover, we have implemented data objects, data-driven gateways, pattern-matching for messages and correlation. Hence, we have produced a complete tool for animating BPMN models in collaborative, multi-instance and data-based contexts.

MIDA inherits its architecture from *bpmn.io*. In fact, MIDA is a web application written in JavaScript that embeds BPMN models into personal browsers without using any server backend. The graphical interface of MIDA, shown in Fig. 9, presents to users a modelling environment. Here, users can create BPMN models using all the facilities of the Camunda modeller. 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, 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, thus also considering data values. The execution of a process instance terminates once all its tokens cannot move forward.

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.,
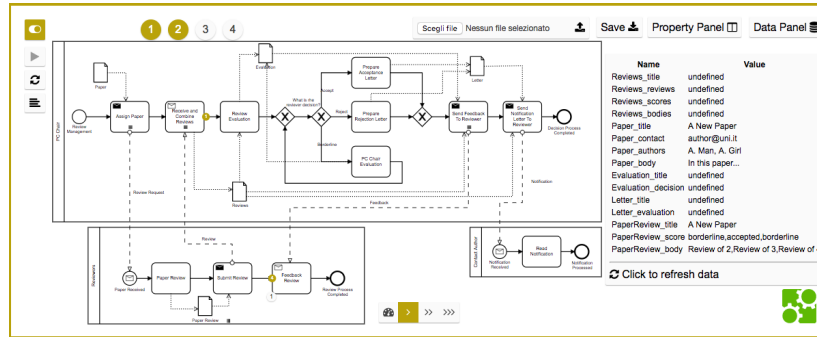
---

[3] `https://github.com/bpmn-io/bpmn-js-token-simulation`

Fig. 9: MIDA web interface.

$\tilde{t_3} = \langle ?\mathsf{Feedback.reviewerName}, ?\mathsf{Feedback.title}, ?\mathsf{Feedback.evaluation} \rangle)$, the feedback messages could not be properly delivered. Indeed, each *Reviewer* instance would be able to match any feedback message, regardless the reviewer name 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 on detecting 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 the current values of data objects, also issues due to bad data handling can be detected using MIDA. For instance, let us suppose that the *Discuss* task in the *PC Chair* pool would not be in a loop, but it would have its outgoing edge directly connected to the XOR join in its right hand side. In this way, after the execution of the *Discuss* task, the task *Send Feedback* would be performed, whereupon the task *Send Results* would be activated. However, the task guard would not be satisfied, because the *Letter* data object would not be properly instantiated. This would cause a deadlock, which can be find out by using MIDA.

The MIDA tool, as well as its source code and examples, is freely available from `http://pros.unicam.it/mida/`.

5 Related Work

In this section discuss related works from the literature focussing on multi-instance collaborations.

5.1 Formalisation of Multiple Instances and Data in the Literature

In the following we refer to the most relevant attempts in formalising multiple instances and data. First, we refer to the works from the BPMN community, since much effort has been devoted to the formalisation of the BPMN standard both for collaboration and choreography diagrams. It is worth noticing that only few works cover multiple instances, and their correlation with the data perspective. Then, we discuss formalisations based on mappings from BPMN to well-known formalisms. Finally, we refer to the wider domain of service-oriented computing.

On Multiple Instances and Data in BPMN collaboration diagrams. In the context of business process management, multiple instances are considered by Meyer et al. in [15]. In this work, process instances and data object instances are linked in order to ensure that two instances of the same process do not have conflicting accesses to their data objects. The correlation mechanism, used in BPMN for the delivering of messages, is instead used here to distinguish and reference data object instances. However, the authors focus on process models where data objects are shared entities (like data stores), and they do not support the exchange of messages between different participants. Use of data object local to instances, exchange of messages between (multi-instantiated) participants, and delivery of messages based on the correlation mechanism are instead the key aspects of collaborations that we focus on. In [16], the authors describe a model-driven approach for BPMN to include the data perspective, enabling the complete automation of data exchange between participants. The challenges they face are data heterogeneity, correlation and 1-to-n communication. Differently from us, the authors do not aim at providing a formal semantics for BPMN multiple instances. Moreover, even if they use data objects in the correlation mechanism, they do not formalise how data can be used in case of data-based decision gateways.

Another interesting work is described in [17], where Kheldoun et al. propose a formal semantics of BPMN covering features such as cancellation, multiple instantiation of sub processes and exception handling, while taking into account the data flow aspect. However, the proposed semantics is not given directly on BPMN but using recursive ECATNets. Moreover, even if they provide a mapping of message flow into RECANet, they do not focus on the collaboration layer. Thus, they do not consider multiple instance pools and do not solve the issue of correlating process instances with messages exchanged between different participants.

Data objects semantics and their use to formalise decision gateways is instead proposed by El-Saber and Boronat in [18]. Differently from us, the authors provide a context-free grammar to formally define guard expressions, while we leave the expression language underspecified, as done in the BPMN standard. Moreover, they impose a model to be well-formed (thus, well-structured), thus restricting the class of BPMN models taken into account. Indeed, we make no assumption on the structure of processes, i.e. we support processes with an *arbitrary topology*. Besides this limitation, there is another drawback concerning BPMN organisational aspects and messages flow. In particular, the formal treatment presented in the paper does not include collaborations and, hence, exchange of messages and multiple instances.

Data perspective in Choreography Diagrams. López et al. [19] propose an automatic detection of the synchronisation points in choreographed models, derived from exchanged objects. They study the choreography problem derived from the synchronisation of multiple instances necessary for the data dependencies. Thus, they do not aim at providing a formal characterisation of BPMN multiple instances and data. Moreover, business processes typically require asynchronous communication, feature that is not supported by the considered choreography diagrams.

Knuplesch et al. [20] 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. In par-

ticular, they focus on interaction models, that in BPMN correspond to choreography diagrams. The flow of message exchanges is specified without having any knowledge about the partner processes, thus data exchanged via messages cannot be used within processes by decision gateways. Finally, their framework does not to support asynchronous message exchange, which is instead common in distributed systems in reality.

Improving data-awareness and data-related capabilities for the modelling and execution of choreographies is the goal of Hahn at al. [21]. They propose a way to unify the data flow across participants with the data flow inside a participant. Anyway, the scope of data objects is global to the overall choreography, while we consider data object scopes local to participant instances, as prescribed by the BPMN standard.

Apart from the specific differences mentioned above, our work differs from the above ones as for the focus on collaboration diagrams, rather than choreographies. This allows us to focalise on multiple process instantiation and correlation of messages, which are not considered by these related works.

On Multiple Instances and Data in other Modelling Languages. Concerning other modelling languages, YAWL provides direct support for the multiple instance patterns [22]. However, YAWL lacks support for handling data. Moreover, modelling the synchronisation of a dynamic number of process instances depending on their termination state cannot be done using the multiple instantiation constructs of YAWL, as the terminated process instances are synchronised according their number and not their states.

In a high-level Petri net, instead, it is possible to use coloured tokens (i.e., tokens with values) to identify multiple instances [23]. However, also in this case there is no explicit support to data handling.

In both cases, process instances are characterised by their identity, rather than by the values of their data, which is however necessary to correlate messages to running instances according to the generic content of the messages. Moreover, even if BPMN can be mapped into these formalisations, reasoning on the semantics of BPMN elements is made more difficult by the fact that such semantics is not directly given in terms of features of the language, but in terms of low-level details of the encodings.

Correlation Management in Service-Oriented Computing. 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 [24] in [1, Sec. 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 [25] 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.

Barros et al. [26] introduce a set of formally defined correlation patterns that provide a basis for evaluating languages and protocols for service implementation in terms of their support for correlation. In the proposed patterns, the correlation mechanism is seen as means for grouping messages into conversations. In our formal treatment we are not specifically interested in identifying conversations, but just correlating messages to process instances in order to properly delivering them.

To sum up, differently from other works, our formalisation is given directly on BPMN, with specific reference to multiple process instances in collaboration diagrams. In particular, we provide a semantics for multiple instance pools, managing asynchronous message exchange and their data-based correlation.

5.2 Business Process Animation in the Literature

It is well known from the literature that BPMN process animators play an important role in understanding the behaviour of business processes. Becker et al. [5] argue that process animation and simulation are essential features for better visualise business processes. More in detail, Oppl and Hoppenbrouwers [27] show how process animation can facilitate the development of modelling skills for non-IT people. The authors also promote animation for validating a business process by adapting its behaviour. This point is also stressed by Desel [6], who shows the importance of using simulations to validate behavioural models of systems. Simulation is mentioned in [28] as a technique that helps in understanding, analysing, and designing processes.

Considering the implementation of a process animator, the faithful correspondence with the semantics is essential. Becker et al. [7] formulate six main quality criteria for business process animation, one of which requires that such process visualisations need to be syntactically and semantically correct. However, Bocciarelli et al. [29] and Cetinkaya et al. [30] point out the existence of a semantic gap between the static process representation and the corresponding dynamic representation.

However, despite the clear need of process animating facilities discussed above, the work on token animation tools, in particular considering BPMN models, is limited [8]. Allweyer and Schweitzer [10] propose a tool which can be used for animating BPMN models. This work covers several BPMN elements and introduces process instances. Anyway, differently from us, that tool animates only processes, as it discards messages exchange, both semantically and graphically. In addition, gateways' decision is performed manually during the animation by users, instead of depending on data. Signavio modeller [31] contains a step-by-step simulator. This animator lets 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. Visual Paradigm [32] provides an animator that supports also collaboration diagrams. This tool allows users to visualise the flow of messages and implements the semantics of receive tasks and events. But, also in this tool, data and multiple instances are missing.

In conclusion, the tools mentioned above do not rely on a precise semantics able to guarantee a correct animation as in our case. Moreover, the missing implementation of process instances and data make impossible to fully animate collaboration diagrams, and to automatically manage decisions on data-based gateways.

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 in BPMN collaborations related to multiple instantiation?

The answer to RQ1 is mainly given in Sec. 3, where we provide a novel operational semantics clarifying the interplay between control features, data, message exchanges and multiple instances. Indeed, in the literature, there is a lack of a formal semantics dealing with all these features at the same time, which is however critical considering the wide adoption of the BPMN specification in distributed scenarios where the same role can be activated multiple times. The answer to RQ2 is instead given in Sec. 4, where we propose MIDA, an animator tool, based on our formal semantics, that provides the visualisation of a collaboration behaviour 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 the assumptions and limitations of our approach, and touching upon directions for future work.

Discussion. Our BPMN operational semantics focusses on the communication mechanisms of collaborative systems, where multiple participants cooperate and share information. Thus, we have intentionally left out those features of BPMN whose formal treatment is orthogonal to the addressed problem, such as timed events and error handling. On the other hand, to keep our formalisation more manageable, multi-instance 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 the 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 AND-Join [33,22], 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. In fact, consider for example the collaboration fragment in Fig. 10(a), where a multi-instance receiving task communicates with a multi-instance pool. Supposing to have three instances of Task A, by applying the mentioned macro replacement we would obtain the collaboration fragment in Fig. 10(b), which is not semantically equivalent. Indeed, each instance of ORG B in Fig. 10(a) has a Task B that sends only one message, while in Fig. 10(b) each instance has a Task B sending three times the same messages, one for each copy of Task A in ORG A. This suggests that multi-instance parallel tasks are not simple macros, but they deserve their own direct formalisation.

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 with 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.
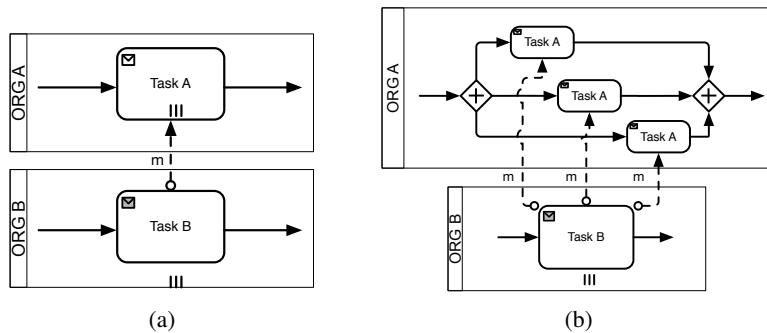
24

Fig. 10: Parallel multi-instance send tasks.

The formalisation would become even more complex if we consider multi-instance sub-processes, which e.g. would require an extension of the correlation mechanism.

Finally, we do not consider data stores, used in BPMN to memorise information that will persist beyond the completion of process instances. Providing a formalisation for data stores would required to extend collaboration configurations with a further data 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 affects different data state functions in the configuration.

Future Work. As a future work, we plan to continue our programme to effectively support modelling and visualisation of BPMN multi-instance collaborations, by overcoming the limitations mentioned above. As a more practical work, we intend to enlarge the range of functionalities provided by MIDA, especially for what concern the data perspective, and improve its usability.

References

1. OMG: Business Process Model and Notation (BPMN V 2.0) (2011)
2. Suchenia et al., A.: Selected approaches towards taxonomy of business process anomalies. In: Advances in Business ICT: New Ideas from Ongoing Research. Springer (2017)
3. Van Gorp, P., Dijkman, R.: A visual token-based formalization of BPMN 2.0 based on in-place transformations. Information and Software Technology **55**(2) (2013) 365–394
4. Börger, E., Thalheim, B.: A Method for Verifiable and Validatable Business Process Modeling. In: Advances in Software Engineering. Volume 5316 of LNCS. Springer (2008) 59–115
5. Hermann et al., A.: Collaborative business process management - A literature-based analysis of methods for supporting model understandability. In: WI. (2017)
6. Desel, J.: Teaching system modeling, simulation and validation. In: WSC. (2000) 1669–1675
7. Becker, J., Kugeler, M., Rosemann, M.: Process management: a guide for the design of business processes. Springer Science & Business Media (2013)
8. Emens, R., Vanderfeesten, I.T.P., Reijers, H.A.: The dynamic visualization of business process models: A prototype and evaluation. In: BPM. Volume 256 of LNBIP., Springer (2016) 559–570
9. Momotko, M., Nowicki, B.: Visualisation of (distributed) process execution based on extended BPMN. In: DEXA, IEEE (2003) 280–284

10. Allweyer, T., Schweitzer, S.: A tool for animating BPMN token flow. In: BPMN Workshop. Volume 125 of LNBIP., Springer (2012) 98–106
11. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007)
12. Corradini et al., F.: On avoiding erroneous synchronization in BPMN processes. In: BIS. Volume 288 of LNBIP., Springer (2017) 106–119
13. Corradini, F., Polini, A., Re, B., Tiezzi, F.: An operational semantics of BPMN collaboration. In: FACS. Volume 9539 of LNCS., Springer (2016) 161–180
14. Corradini, F., Muzi, C., Re, B., Rossi, L., Tiezzi, F.: Global vs. local semantics of BPMN 2.0 or-join. In: SOFSEM. Volume 10706 of LNCS., Springer (2018) 321–336
15. Meyer, A., Pufahl, L., Fahland, D., Weske, M.: Modeling and enacting complex data dependencies in business processes. In: BPM. Volume 8094 of LNCS., Springer (2013) 171–186
16. Meyer et al., A.: Data perspective in process choreographies: modeling and execution. In: Techn. Ber. BPM Center Report BPM-13-29. BPMcenter. org. (2013)
17. Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal verification of complex business processes based on high-level Petri nets. Information Sciences **385-386** (2017) 39–54
18. El-Saber, N.A.: CMMI-CM compliance checking of formal BPMN models using Maude. PhD thesis, Department of Computer Science (2015)
19. López et al., M.T.G.: Guiding the creation of choreographed processes with multiple instances based on data models. In: BPMWorkshops. Volume 281 of LNBIP. (2016) 239–251
20. Knuplesch, D., Pryss, R., Reichert, M.: Data-aware interaction in distributed and collaborative workflows: modeling, semantics, correctness. In: CollaborateCom, IEEE (2012) 223–232
21. Hahn, M., Breitenbücher, U., Kopp, O., Leymann, F.: Modeling and execution of data-aware choreographies: an overview. Computer Science-Research and Development (2017) 1–12
22. Wohed, P., van der Aalst, W.M., Dumas, M., ter Hofstede, A.H., Russell, N.: Pattern-based analysis of UML activity diagrams. Beta, Research School for Operations Management and Logistics, Eindhoven (2004)
23. Van Der Aalst, W.M., Ter Hofstede, A.H.: YAWL: yet another workflow language. Information systems **30**(4) (2005) 245–275
24. OASIS WSBPEL TC: Web Services Business Process Execution Language Version 2.0. Technical report, OASIS (April 2007)
25. Pugliese, R., Tiezzi, F.: A calculus for orchestration of web services. Journal of Applied Logic **10**(1) (2012) 2–31
26. Barros, A.P., Decker, G., Dumas, M., Weber, F.: Correlation patterns in service-oriented architectures. In: FASE. Volume 4422 of LNCS., Springer (2007) 245–259
27. Oppl, S., Hoppenbrouwers, S.: Introducing fundamental concepts of process modeling through participatory simulation. In: CAiSE. Volume 286 of LNBIP., Springer (2017) 110–122
28. Kettinger, W.J., Teng, J.T., Guha, S.: Business process change: a study of methodologies, techniques, and tools. MIS quarterly (1997) 55–80
29. Bocciarelli et al., P.: A transformation approach to enact the design-time simulation of bpmn models. In: WETICE, IEEE (2014) 199–204
30. Çetinkaya, D., Verbraeck, A., Seck, M.D.: Model continuity in discrete event simulation: A framework for model-driven development of simulation models. ACM TOMACS **25**(3) (2015) 17
31. Signavio GmbH: Signavio (2018) `http://www.signavio.com/`.
32. Visual Paradigm: Visual paradigm (2018) `https://www.visual-paradigm.com/`.
33. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology **50**(12) (2008) 1281–1294