Università degli Studi di Camerino

School of Advanced Studies

Dottorato di Ricerca in Scienze e Tecnologie
Computer Science - XXXII Ciclo

Formalization and Animation of Business Process Collaborations

Relatore

Prof. Francesco Tiezzi

Co-Relatore

Prof.ssa Barbara Re

Commissione Esaminatrice

Prof. Maurice ter Beek

Prof. Pascal Poizat

Dottorando

Lorenzo Rossi

Anno Accademico 2019-2020

University of Camerino

School of Advanced Studies

Doctor of Philosophy in Sciences and Technology

Computer Science - XXXII Cycle

Formalization and Animation of Business Process Collaborations

Supervisor

Prof. Francesco Tiezzi

Co-Supervisor

Prof.ssa Barbara Re

Doctoral Examination Committee

Prof. Maurice ter Beek

Prof. Pascal Poizat

PhD Candidate

Lorenzo Rossi

Academic Year 2019-2020

To Chi:

I'm still convinced that you're the horse to bet on.

ABSTRACT OF THE DISSERTATION

The increasing adoption of modeling methods contributes to the better understanding of the flow of processes from the internal behavior of a single organization to a wider perspective where several organizations exchange messages. In this regard, the BPMN standard provides a suitable abstraction for modeling, analyzing, executing, and monitoring business processes within the same organization or involving multiple ones.

Even if this is a widely accepted notation, a major drawback of BPMN is related to the complexity of the semi-formal definition of its meta-model. Moreover, only limited efforts have been expended in formalizing its execution semantics, especially for what concerns the interplay among control features, data handling and exchange of messages in scenarios requiring multiple instances of interacting participants. Consequently, the modeling of a BPMN diagram, as well as fully understanding of its behavior, may result very difficult in presence of such concepts.

Thus, providing a formal semantics of the BPMN notation is crucial for shaping collaborative systems and guaranteeing that these systems behave as they are supposed to. Moreover, figuring out the interplay between control flow, messages, and data, by looking at static process models, is in general error-prone and time-consuming. In this regard, visualization techniques, such as animation, can effectively support the designer. In addition, a formal semantics gives the possibility to simulate business processes.

This thesis faces these problems by providing a formal semantics for BPMN collaborations that includes elements dealing with multiple instances, i.e., multi-instance pools and sequential/parallel multi-instance tasks. Beyond defining a novel formalization, the thesis proposes solutions that exploit this formalization to dynamically visualize the behavior and the business context of BPMN diagrams through 2D and 3D animation, and for generating

event logs from the simulation of business processes.

LIST OF PUBLICATIONS

- F. Corradini, A. Morichetta, A. Polini, B. Re, L. Rossi, F. Tiezzi. *Correctness checking for BPMN collaborations with sub-processes.* Journal of Systems and Software, 2020. 110594.

- F. Corradini, C. Muzi, B. Re, L. Rossi, and F. Tiezzi. *Formalising and Animating Multiple Instances in BPMN Collaborations.* Information Systems, 2019. 101459.

- B. M. Abdul, F. Corradini, B. Re, L. Rossi, F. Tiezzi. *UBBA: Unity Based BPMN Animator.* In: International Conference on Advanced Information Systems Engineering, vol. 350 of LNBIP, pages 1-9. Springer, 2019.

- C. Muzi, L. Pufahl, L. Rossi, M. Weske, and F. Tiezzi. *Formalising BPMN Service Interaction Patterns.* In: The Practice of Enterprise Modeling, vol. 335 of LNBIP, pages 3-20. Springer, 2018.

- F. Corradini, C. Muzi, B. Re, L. Rossi, and F. Tiezzi. *Animating Multiple Instances in BPMN Collaborations: from Formal Semantics to Tool Support.* In: International Conference on Business Process Management, vol. 11080 of LNCS, pages 83-101. Springer, 2018.

- F. Corradini, C. Muzi, B. Re, L. Rossi, and F. Tiezzi. *MIDA: Multiple Instances and Data Animator.* In: Online Proceedings of BPM Demo Track. pages 86-90. CEUR-WS.org, 2018.

- F. Corradini, C. Muzi, B. Re, L. Rossi, and F. Tiezzi. *Global vs. local semantics of BPMN 2.0 OR-join.* In: International Conference on Current Trends in Theory and Practice of Informatics, vol. 10706 of LNCS, pages 321-336. Springer, 2017.

- F. Corradini, A. Polini, B. Re, L. Rossi, and F. Tiezzi. *Supporting Multi-layer Modeling in BPMN Collaborations.* In: Workshop on

Enterprise and Organizational Modeling and Simulation, vol. 298 of LNBIP, pages 53-67. Springer, 2017.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

PART I

INTRODUCTION AND BACKGROUND

CHAPTER 1

INTRODUCTION

In the last years, the Business Process Model and Notation (BPMN) [53] became the most prominent notation for representing business processes, thanks to its wide usage in academic and industrial contexts. BPMN is a well-established standard for describing single organization processes and their compositions in a very intuitive way, by means of *Process* and *Collaboration* diagrams respectively. These allow stakeholders, i.e., knowledge workers with their organizational roles and skills [79], to communicate and reason about business processes in a standard manner. Usually, companies do not act alone, but often interact with other organizations in order to reach a shared goal. Therefore, it is interesting to consider collaborative systems where participants can interact and share information and data. BPMN collaboration diagrams are particularly suitable to describe this kind of scenarios, while BPMN process diagrams focus on single participants. However, this demands for a clear understanding of the internal behaviors and the interactions among participants. To ensure proper carrying out of such interactions, each participant's process should be provided with enough information about these interactions, i.e., messages they must or may send and receive in a given context. This is particularly important when multiple instances of interacting participants are involved. In this regard, BPMN collaboration diagrams result to be an effective way to reflect how multiple participants cooperate.

The rest of the chapter introduces the motivations behind the research presented in this thesis, and the research questions it aims to answer. Finally, the thesis structure is provided.

1.1 Motivations

The effective and efficient handling of business processes is a primary goal of organizations in order to conduct a successful business. However, the modeling of organizational processes is not always an easy task for designers, especially if they include different participants, and make use of communication to spread data. Therefore, reflecting in a process model what organizations are supposed to do is a time-consuming activity that requires one to be familiar with the semantics of the process elements.

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 [64]. This becomes a more prominent issue if one considers BPMN supporting tools, such as animators and simulators, whose implementation of the execution semantics may not be compliant with the standard and be different from each other, thus undermining models portability and tools effectiveness [4].

To overcome these issues, several formalizations have been proposed, mainly focusing on the process diagrams (e.g., [21, 18, 16, 25, 23, 81, 8, 73]). Less attention has been paid to provide a formal semantics for collaboration diagrams capturing the interplay between control features, message exchanges, and data. These aspects are strongly related, especially when multi-instance participants have to interact. 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, data 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 yet been proposed in the literature. Therefore, the absence of a comprehensive semantic framework for BPMN reduces the models comprehension.

Considering the representation of BPMN diagrams, the use of *static* 2D flow charts, where the graphics of the elements embeds their semantic categories (e.g., rectangles for activities, diamonds for choices, circles for events) affects the models understandability. Indeed, even if BPMN has a graphical representation that is very straightforward to experts, it may result difficult to understand for those stakeholders [9] who are not familiar with the syntax and semantics of BPMN. Furthermore, when facing very large process and collaboration diagrams it results difficult to follow their execution semantics, while the use of *static* flow chart representations limits the amount of information the user can perceive [78]. In this regard, the literature fosters new process and collaboration diagrams visualization techniques, such as *animation* in 2D and 3D environments, capable of improving the use of the BPMN notation. On the one hand, the animation of process and collaboration diagrams can increase their understanding [33, 24, 5] and also the possibility to debug them, showing into practice the behavior of the process model. On the other hand, a virtual world representing a process and a collaboration diagram can enhance the communication activities, thus facilitating interactions between designers and the stakeholders [30].

1.2 Research Objectives

This thesis aims at answering two research questions that are strictly related one to the other. The first objective is to provide a formal characterization of the BPMN semantics specifically given for collaboration models, with the aim of fostering a correct modeling and understanding of the BPMN collaborations. The goal is to present a formal framework able to capture the different aspects of the BPMN notation, with a specific focus on *multiple instantiation* of processes, *data* handling, and *message* exchange.

Secondly, by means of this formalization, this thesis points at developing *animation* and *simulation* tools to help both designers and stakeholders. These goals can be achieved by answering the following research questions.

RQ1

What is the precise semantics of multi-instance BPMN collaborations in presence of data and message exchanges?

RQ2

Can a formal semantics for multi-instance BPMN collaborations drive the development of software tools based on BPMN collaboration diagrams?

To answer RQ1, an **operational semantics of BPMN collaboration models is provided, including multi-instance elements and taking into account the data perspective.** To answer RQ2, this work goes beyond the mere formalization, by developing **tools that faithfully implement the proposed formal semantics in order to animate or simulate the execution of multi-instance collaborations including data**.

1.3 Thesis Structure

This thesis is organized into four parts and seven chapters, as shown in Figure 1.1. Following it is provided a brief overview on the rest of the thesis.

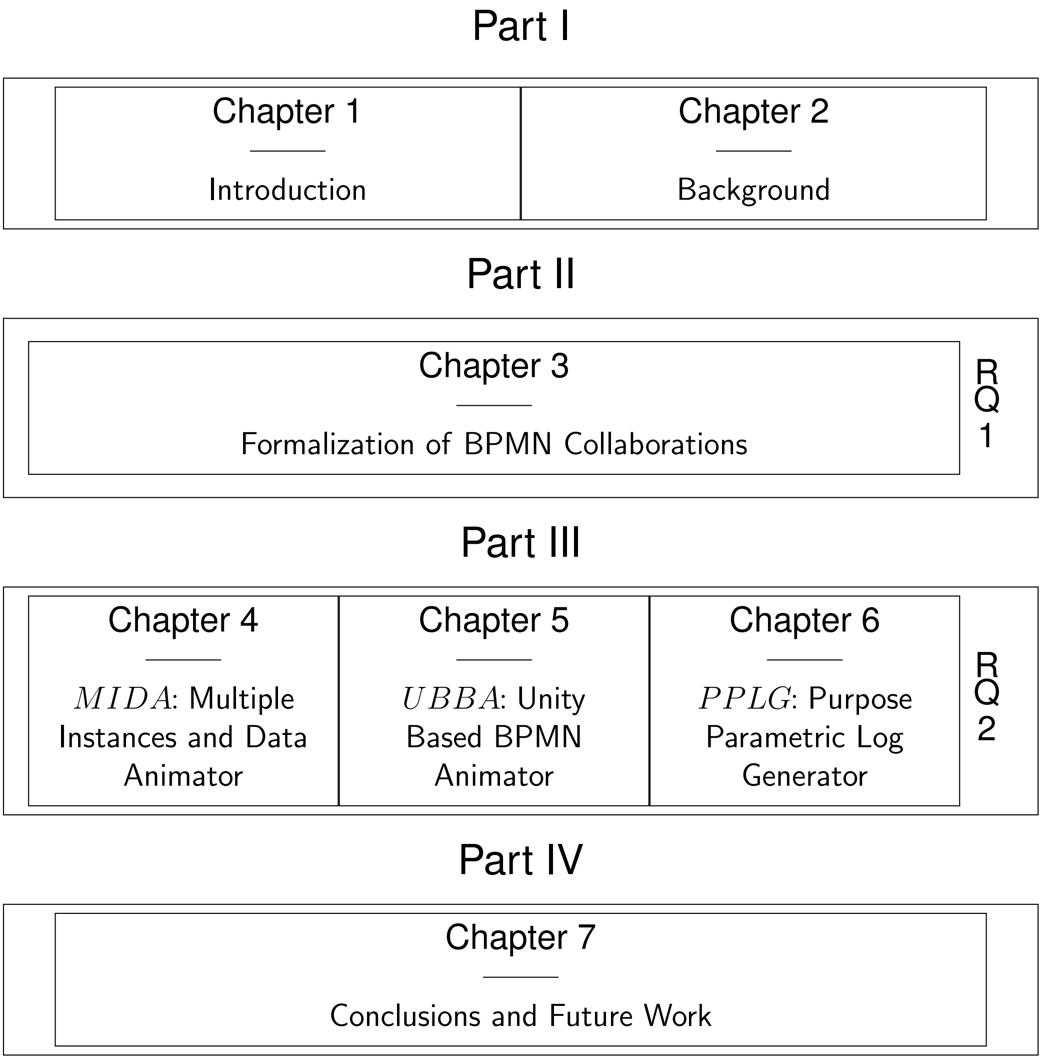

Part I
Chapter 1
Introduction
Chapter 2
Background
Part II
Chapter 3
Formalization of BPMN Collaborations
RQ1
Part III
Chapter 4
MIDA: Multiple Instances and Data Animator
Chapter 5
UBBA: Unity Based BPMN Animator
Chapter 6
PPLG: Purpose Parametric Log Generator
RQ2
Part IV
Chapter 7
Conclusions and Future Work

Figure 1.1: Thesis structure.

Part I - Introduction and Background. This first Part of the thesis explains the intent of this work and the fundamental concepts.

- **Chapter 2 - Background.** This chapter collects all the background materials and concepts, mainly regarding *business process management*, needed for a proper understanding of the thesis.

Part II - BPMN Formal Framework. This Part is composed by a single chapter that regards the formalization of BPMN.

- **Chapter 3 - Formalization of BPMN Collaborations.** This chapter provides the whole formal framework defined for BPMN collaboration diagrams in presence of multiple-instances, data, and message exchanges. It discusses the most tricky parts of the standard and formalizes syntax and semantics. A comparison of the given formalization with existing approaches in the literature is also presented.

Part III - Formalization at Work. This Part shows the application of the proposed formalization.

- **Chapter 4 - Multiple Instances and Data Animator.** This chapter introduces the animation of business processes and presents a 2D animator called MIDA, and shows the benefits deriving from it.
- **Chapter 5 - Unity Based BPMN Animator.** This chapter present an alternative approach for animating process models based on 3D virtual worlds.
- **Chapter 6 - Purpose Parametric Log Generator.** This chapter presents an automated methodology for the generation, through simulation, of event logs suitable for different mining purposes.

Part IV - Conclusions and Future Work. This Part concludes the work.

- **Chapter 7 - Conclusions and Future Work.** This chapter summarizes the work done and presents areas and topics that could be investigated in the future.

CHAPTER 2

BACKGROUND

This chapter aims at gently introducing the set of background notions needed for fully understanding the dissertation content. Firstly, it presents the business process management and the phases composing the business process life-cycle (BP life-cycle), with a focus on the concepts relevant for this thesis. Subsequently, the chapter introduces the standard modeling notation BPMN 2.0 and the running example consisting in a BPMN scenario from where collaboration diagrams are extrapolated to be exploited thorough the rest of the thesis. To conclude, relevant notions about formal methods techniques are given, focusing on the process of model formalization via an operational semantics.

2.1 Business Process Management

Business Process Management (BPM) is the set of all activities to support and improve organization performance by managing chains of events, tasks, and decisions that ultimately add value to the organization. It includes concepts, methods, and techniques to support the modeling, the analysis, the execution, and the monitoring of business processes [79].

A Business Process is described as "*a collection of related and structured activities undertaken by one or more organizations in order to pursue some particular goal. Within an organization a business process results in the provisioning of services or in the production of goods for internal or external stakeholders. Moreover business processes are often interrelated since the execution of a business process often results in the activation of related business processes within the same or other organizations*" [44].

Business processes derive from the observation of products or services provided by a company. Business processes are the outcome of a number of activities, and are the key instrument for organizing these activities and improving the understanding of their interrelationships.

Moreover, there exist business process activities that can be enacted automatically by information systems, without any human involvement. Hence, business processes are an important concept to facilitating the collaboration between companies and information systems. More and more business processes also play an important role in the design and realization of flexible information systems. These information systems are essential for providing the technical basis useful for a quick implementation of new functionalities that realize new products or services.

BPM is characterized by a set of steps that occur cyclically in order to adapt and improve the model. Hence, BPM involves a continuous cycle, see Figure 2.1, comprising the following phases: *modeling*, *analysis*, *execution*, and *monitoring*.

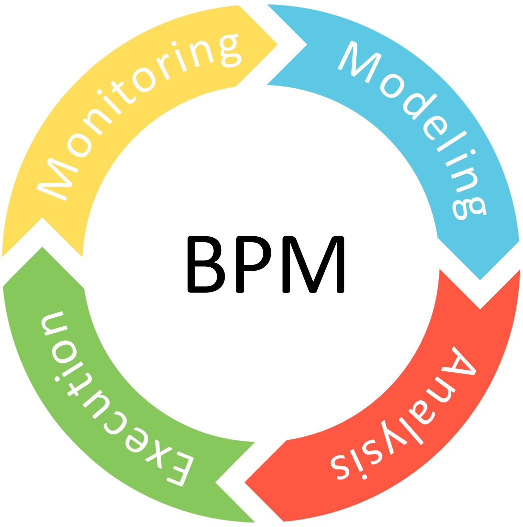


Figure 2.1: BP life-cycle.

- **Process modeling**. Business process models are important at various stages. The modeling phase identifies the business goals and the involved stakeholders in the process. During this phase, the business process model is drawn by a business process designer. Since process models are meant to facilitate communication between stakeholders, they should be easy to understand. To this aim, modeling languages are used to design business process models, so that different stakeholders can efficiently communicate, refine and improve the models. The outcome of process modeling is a new or updated process architecture that provides an overall view of the processes in an organization and their relationships. Here, the current state of each of the relevant processes is provided with one or several as-is process models. This phase is also called *as-is process modeling*.

- **Process analysis**. It uses the *as-is* process model as reference for verifying if it works seamlessly. The goal is to identify changes to the process that would help to address the issues identified in the previous phase and allow the organization to meet its performance objectives. Analysis can be done to detect syntactic, structural or behavioral problems. This can be done by means of different techniques such as animation, verification, etc. One of its aims is to ensure correctness of the designed business process model, because erroneous behavior can cause high costs for the involved organizations and damage their reputation.

- **Process execution**. Weaknesses spotted during the analysis phase are fixed to produce a process model to execute. In the execution phase, the business processes are enacted for achieving the business goals. The execution can be performed manually by humans, automatically by means of software solutions, or with a combination of both methods.

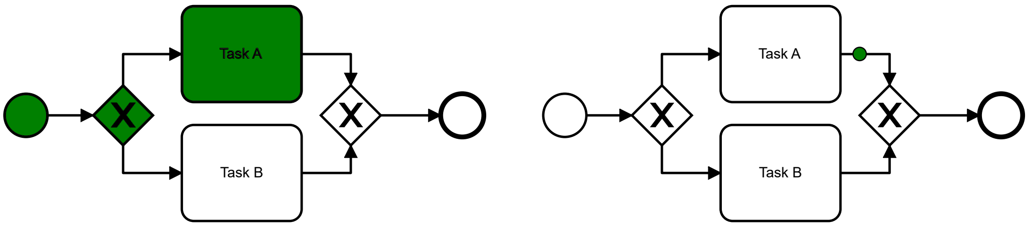
- **Process monitoring**. To close the life-cycle, the monitoring phase constantly oversees the process in execution to collect insights. This information allows to evaluate the outcomes of the business process, and understand how much is required to reach the business goals. Among the other approaches, this phase is mainly carried on by process mining techniques.

This thesis tackles *modeling*, *analysis*, and *monitoring* phases. In particular, it first faces the modeling phase via the formalization of the BPMN semantics. Then, it exploits animation techniques to enhance the analysis of process models. Finally, it investigates the monitoring phase defining a novel methodology that generates event logs for process mining through simulation.

2.1.1 Process Animation

Among the different techniques helping the analysis of business models, the ones based on visualization features can be used to enhance process model comprehension, and spot modeling errors. These techniques are exploited in novel ways (e.g., animations, games, virtual words) to turn static representations into dynamic ones [3].

Animation consists in visualizing the model execution through graphic effects. Basically, there exist two methods for animating a process model: one consists in highlighting the process elements that have been already executed, like in Figure 2.2 (a), the other one uses tokens crossing the model, where the moving of the token represents the shifts between configuration to another one [30], as in Figure 2.2 (b).



(a) (b)

Figure 2.2: Process animation: highlighting (a) vs. token flow (b).

Animation can enhance the understanding of business processes behavior [33, 24], especially in the presence of a faithful correspondence with a precise semantics [5]. Moreover, despite the fact that business process notations are usually quite intuitive, their way of representing processes with 2D flow charts may result effective just for experts. In this regard, there exists techniques for animating business processes in 3D. The aim is to avoid complex process modeling notation in favor of 3D representations that better embody the situation described by the business process. Indeed, business process notations are quite easy to understand for business experts, but they can result incomprehensible for stakeholders.

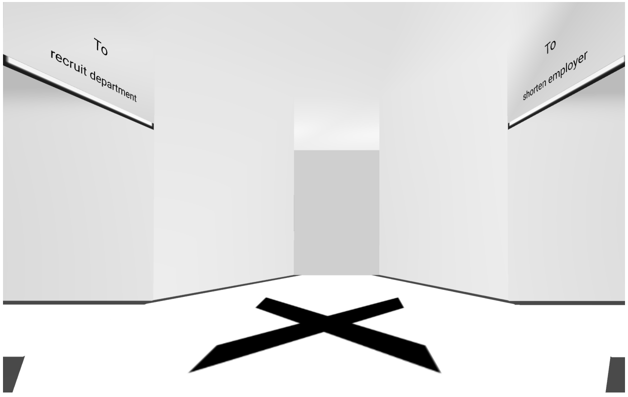


Figure 2.3: 3D process animation example.

2.1.2 Process Mining

Thanks to the wide usage of information systems, nowadays, process mining [67] is recognized as an important discipline for extracting useful information from the execution of business processes. Indeed, process mining aims at automatically providing insights on the process execution. This recently discovered technique helps in closing the loop of the BP life-cycle, Figure 2.1. Indeed, the *monitoring* phase uses data coming from the process execution in order to derive additional information regarding problems or possible improvements.

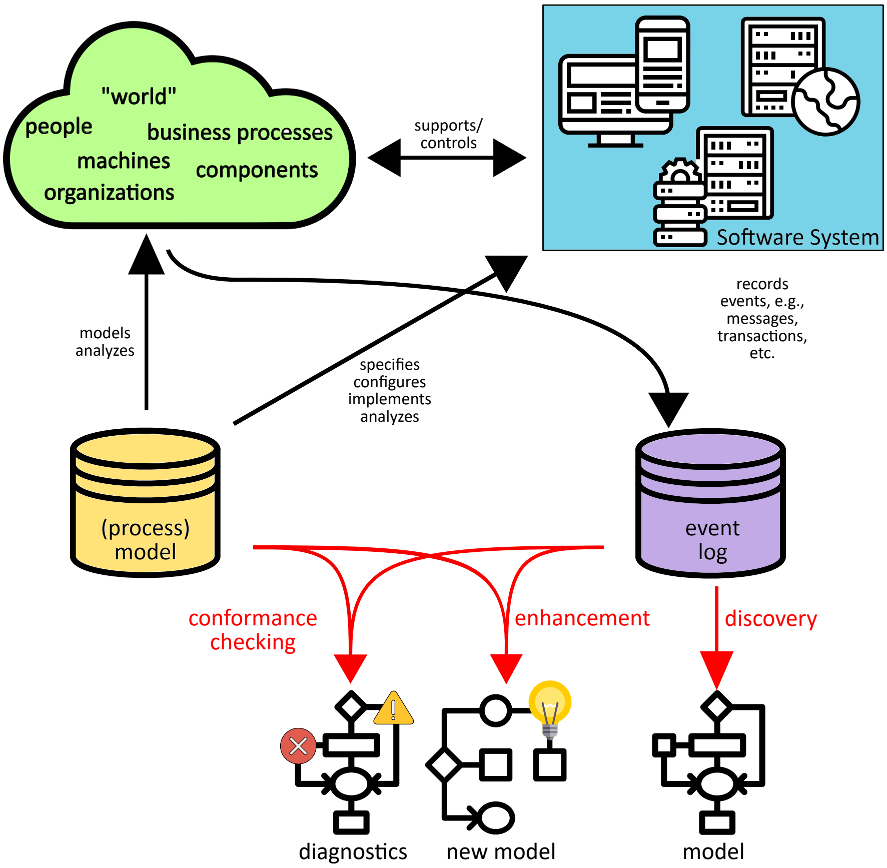


Figure 2.4: Mining techniques [68].

Within the sphere of process mining exists several purposes supported by techniques for analyzing business processes from the event logs, Figure 2.4.

- ***Process discovery.*** Process discovery is the most common and also most challenging activity in the process mining area. This technique permits to recreate a process model from the analysis of its own event log. The reliability of the result (hence the similarity between the original and the mined model) depends on the log completeness and on the used discovery algorithm.

- ***Conformance checking.*** Conformance checking compares the event log with the activities in the process model to discover in the event log deviations and differences with the intent of the process model.

Conformance checking can detect unwanted deviations in the process model execution that can refer to frauds or implementation issues.

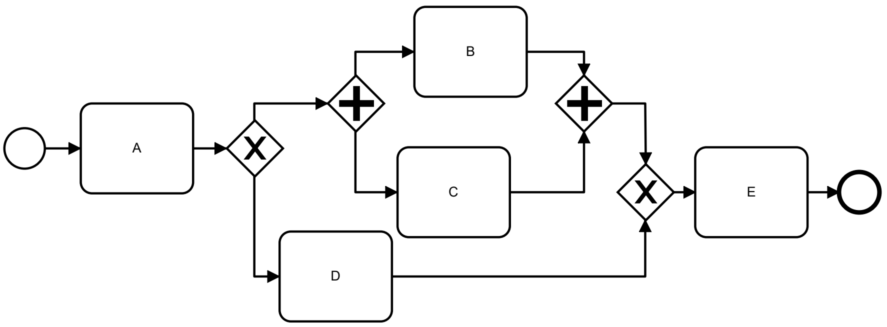
- ***Enhancement***. The enhancement points at improving process models using insights discovered into the event logs. This technique changes the original process in two ways. One is the repair, by which deviation or unwanted behavior can be fixed. The other one is the extension, that consists in adding new information into the process model that comes from logs.

Event logs are the first class citizens in process mining. They are the sources of knowledge where to apply process mining approaches. Event logs store data about the performing of activities that were recorded by information systems. They consist of sequences of records, each of which refers to a performed process activity.

Data retrieved form the execution of a process can be used to increase the knowledge on it. Indeed, events embody attributes that describe additional information about them. Usually these attributes are the activity name, and the execution timestamp; however, depending on the information system generating the log, attributes can also refer to different perspectives of the process model (e.g., control-flow, resource, information).

More practically, an event log can be seen as a table where each row is an event, and the columns list the attributes which may be present or not. For instance, Figure 2.5 shows a process model and a portion of an event log generated by the process execution. The reference model is usually called the *gold standard* [10], a ground truth acting as a compass during the mining procedures. Focusing on the table, it has four columns, some of those, *Case* and *Event*, can be used to join events into a *trace*, or to uniquely identify them. The last two columns, *Timestamp* and *Activity*, contain information specific of the activity.

In addition to these four attributes, the taxonomy of the events can be enriched at will with information regarding different perspectives of the activity execution. At this regard, one can exploit the Extensible Event Stream (XES) [36]. This is a standard for representing event logs à la XML. It is widely recognized as a de-facto standard and is used by many applications and analysis tools. By following the syntax and semantics of XES, analysis tools can easily interpret event data from applications. The XES standard consists of three hierarchical levels: log, trace and events. Each of these levels can contain attributes to add information to the respective level.



Case	Event	Timestamp	Activity	...
1	0001	30-12-2019:11.02	A	...
1	0002	30-12-2019:11.31	D	...
1	0003	30-12-2019:11.44	E	...
2	0004	31-12-2019:01.25	A	...
2	0005	31-12-2019:01.43	B	...
2	0006	31-12-2019:02.09	C	...
2	0007	31-12-2019:03.02	E	...
...	...	...	...	...

Figure 2.5: Event log example.

2.2 Business Process Model and Notation 2.0

Many different languages and graphical notations have been proposed to represent business process models, differing both in the possibility to express aspects related to the organization perspectives and in the level of formality used to define the elements composing the notation. BPMN 2.0 [53] is currently acquiring a clear predominance. It has been standardized by the Object Management Group (OMG) and it is now widely accepted both in industry and academia. Its first goal is to provide a notation that is readily understandable by all business users. This includes the business analysts that create the initial drafts of the processes to the technical developers responsible for implementing the technology that will perform those processes.

BPMN's success comes from its versatility and capability to represent business processes with different levels of detail and for different purposes. Business process models are expressed in business process diagrams. Each diagram consists of a set of modeling elements. These elements are partitioned into a core element set and a complete element set. In particular, the BPMN notation allows to design different kinds of diagrams: process, collaboration, choreography and conversation diagrams. Moreover, in deriving a business process model, many different information and perspectives of an organization can be captured [58]. This thesis shows the capability of BPMN to describe the following aspects: who should perform the activities (organization perspective), when they should be performed and how they are organized in a flow (control-flow perspective) and, finally, which data is needed and produced (information perspective), see Figure 2.6. For a complete and detailed description of each BPMN diagram, please refer to the official BPMN Specification [53]. The thesis focuses on process diagrams, which are used to represent processes within a single organization, and on collaboration diagrams, that model processes of different organizations exchanging messages and cooperating to reach a shared business goal.

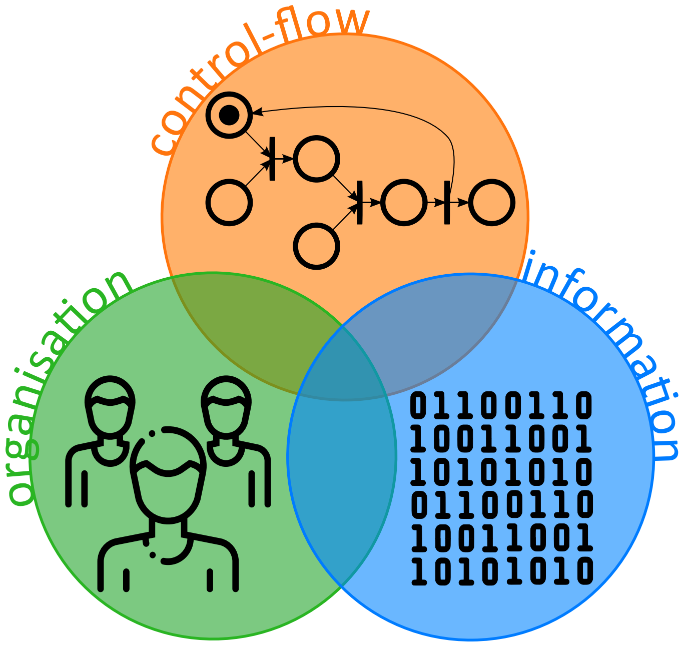


Figure 2.6: Business process perspectives.

The next sections describe the BPMN elements that will be considered in the thesis, some basic definitions regarding process models, and then introduce an example used to show the reader how to employ the BPMN notation and throughout the work as a running example to show the thesis results.

2.2.1 BPMN Notation

This section illustrates the BPMN elements considered in this thesis, which include: pools, tasks, events, gateway, connecting edges and data objects.

- **Pools**, Figure 2.7, are used to represent participants or organizations involved in the collaboration, and include details on internal process specifications and related elements. Pools are drawn as rectangles, and they usually have a name associated, referring to the name of the organization. BPMN allows to assign a multi-instance marker (three vertical lines) to a pool, representing multiple instances playing the same role.

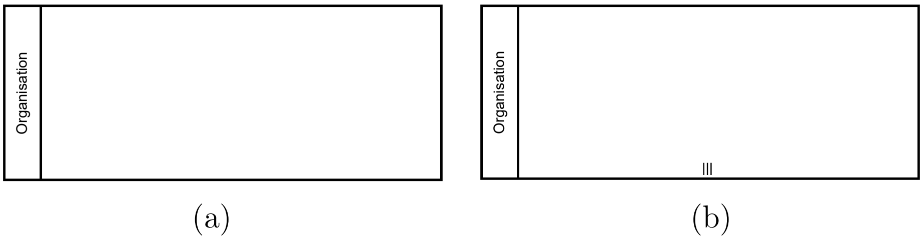


Figure 2.7: A BPMN pool (a) and a BPMN multi-instance pool (b).

- **Tasks**, Figure 2.8, are used to represent a specific piece of work to be performed within a process. They are drawn as rectangles with rounded corners.

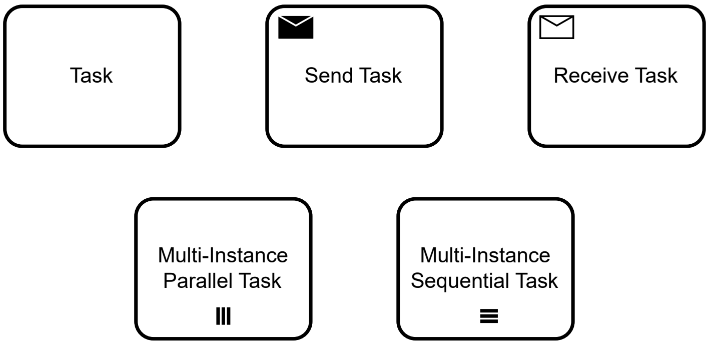


Figure 2.8: Considered BPMN activities.

  - *Task* is a simple task that represents the performing of an activity.
  - *Send Task* is a task that represents the performing of an activity involving the sending of a message.
  - *Receive Task* is a task that represents the performing of an activity involving the receiving of a message.
  - *Multi-instance Parallel Task* is a task that represents the performing of an activity more than one time concurrently.
  - *Multi-instance Sequential Task* is a task that represents the performing of an activity more than one time one after the other.

- **Connecting Edges**, Figure 2.9, are used to connect process elements inside or across different pools. *Sequence Edges* are solid connectors used to specify the internal flow of the process, thus ordering elements in the same pool, while *Message Edges* are dashed connectors used to visualize communication flows between organizations[1].

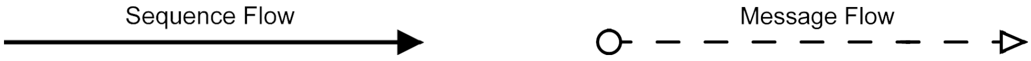


Figure 2.9: BPMN connecting edges.

- **Events**, Figure 2.10, are used to represent something that can happen. An event can be a *Start Event* representing the point from which a process starts, an *Intermediate Event* representing something that happens during process execution, or an *End Event* representing the process termination. Events are drawn as circles. When an event is source or target of a message edge, it is called *Message Event*. According to the different kinds of message edge connections, it is possible to distinguish between the following type of events.

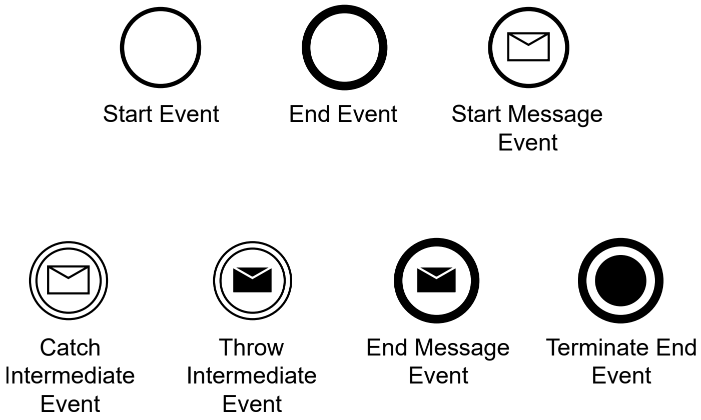


Figure 2.10: Considered BPMN events.

  - *Start Message Event* is a start event with an incoming message edge; the event element catches a message and starts a process.
  - *Catch Intermediate Event* is an intermediate event with an incoming message edge; the event element receives a message.
  - *Throw Intermediate Event* is an intermediate event with an outgoing message edge; the event element sends a message.
  - *End Message Event* is an end event with an outgoing message edge; the event element sends a message and ends a process.

[1] As a matter of terminology, Sequence Edge and Message Edge are referred in the BPMN specification as Sequence Flow and Message Flow, respectively.

There is also a particular type of end event, the *Terminate End Event*, displayed by a thick circle with a darkened circle inside; it stops and aborts the running process - all the ongoing activities are aborted and the process is abnormally terminated.

- **Gateways**, Figure 2.11, are used to manage the flow of a process both for parallel activities and choices. Gateways are drawn as diamonds and act as either join nodes (merging incoming sequence edges) or split nodes (forking into outgoing sequence edges). It is possible also to express gateways with multiple incoming and multiple outgoing edges in BPMN. These gateways are called mixed gateways. Since two behaviors (split and join) are expressed by a single concept, best practice is not to use mixed gateways but to use a sequence of two gateways with the respective split and join behavior instead. Different types of gateways are available.

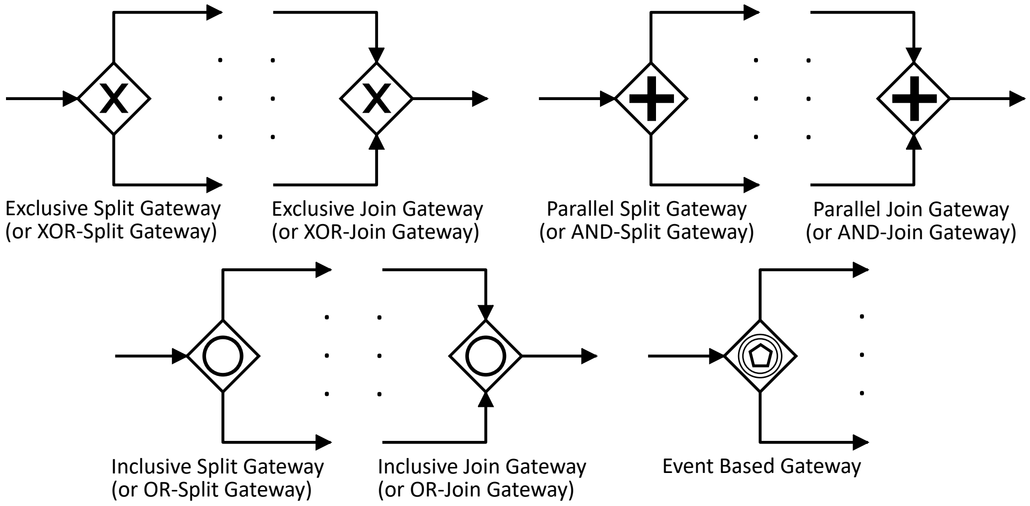


Figure 2.11: Considered BPMN gateways.

  - An *Exclusive Gateway* (or *XOR gateway*) gives the possibility to describe choices. In particular, a XOR-Split gateway is used after a decision to fork the flow into branches. When executed, it activates exactly one outgoing edge. A XOR-Join gateway acts as a pass-through, meaning that it is activated each time the gateway is reached. A XOR gateway is drawn with a diamond marked with the symbol "×".
  - A *Parallel Gateway* (or *AND gateway*) enables parallel execution flows. An AND-Split gateway is used to model the parallel execution of two or more branches, as all outgoing sequence edges are activated simultaneously. An AND-Join gateway synchronizes

the execution of two or more parallel branches, as it waits for all incoming sequence edges to complete before triggering the outgoing flow. An AND gateway is drawn with a diamond marked with the symbol "+" .

  - An *Inclusive Gateway* (or *OR gateway*) gives the possibility to select an arbitrary number of (parallel) flows. In fact, an OR-Split gateway is similar to the XOR-Split one, but its outgoing branches do not need to be mutually exclusive. An OR-Join gateway synchronizes the execution of two or more parallel branches, as it waits for all *active* incoming branches to complete before triggering the outgoing flow. An OR gateway is drawn with a diamond marked with the symbol "○".
  - An *Event-Based gateway* is used after a decision to fork the flow into branches according to external choice. Its outgoing branches activation depends on the occurrence of catching events. Basically, such events are in a race condition, where the first event that is triggered wins and disables the other ones. An event-based gateway is drawn with a diamond marked with the symbol "◎" double rounded.

  Notably, XOR and OR splitting gateways may have guard conditions in their outgoing sequence edges. Thus in some cases the decision on XOR/OR-Split gateways is taken non-deterministically and in other cases conditions are used to decide which edge to activate according to data values.

- **Artifacts**, Figure 2.12, are used to show additional information about a business process that "is not directly relevant for sequence flow or message flow of the process", as the standard mentions. Each artifact can be associated with flow elements. There are different types of artifacts. Among these, the thesis takes into account data objects.

  Data objects represent information and material flowing in and out of activities. They are depicted as a document with the upper-right corner folded over, and linked to activities with a dotted arrow with an open arrowhead (called *data association* in BPMN). The direction of the data association is used to establish whether a data object is an input or output for a given activity. Paper documents and electronic documents, as well as information on any type of medium can be represented by data objects. Sometimes data objects can refer to a state. Indicating data objects' states is optional: it can be done by appending the name of the state between square brackets to a data object's label. Different types of data objects are available, they are reported in the following.

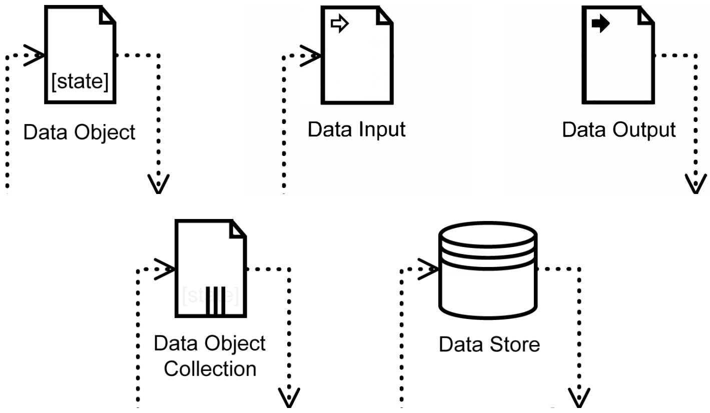


Figure 2.12: BPMN types of data objects.

- The *Data Input* element is used to express (external) input data, and it can be read by an activity.
- The *Data Output* element is used to express output data, and it can be generated by an activity.
- The *Data Object Collection* element refers to multi-instance data objects; using this a designer can express the involvement of more than one Data Object.
- The *Data Store* element is used to express data that persists after the process instance finishes.

As stated in the BPMN standard, a key concept related to the BPMN process execution refers to the notion of *token*. The BPMN standard states that

"a token is a theoretical concept that is used as an aid to define the behavior of a process that is being performed" [53, Sec. 7.1.1].

A token is commonly generated by a start event, traverses the sequence edges and it is eventually consumed by and end event. Process elements retrieve one or more tokens from the incoming sequence flow to be executed. Once finished, they may produce one or more tokens on the outgoing sequence flows, depending on their behavior. In the collaboration, the process execution also triggers message flows able to generate messages. They will be referred to message flow tokens.

For better managing the semantics of the BPMN notation, sometimes it is necessary to query the topology of the diagram. In order to do that, one may consider a process model as a *direct graph* transforming sequence flows and message flows into arcs, and flow elements into vertices. More precisely, it is defined as follows, $G = (V, \mathbb{E}, A)$ where:

- V is a set of *vertices*, ranged over by v and consisting of start events, end events, and gateways; and
- A is a set of *arrows*, consisting of triples (v_1, e, v_2) with $v_1, v_2 \in V$, $v_1 \neq v_2$ and $\mathsf{e} \in \mathbb{E}$, where $\mathbb{E}$ is the set of all (sequence) edges.

Since edges are uniquely identified in a BPMN model, it results that for each (v_1, e, v_2) in A there exists no triple $(v_1', \mathsf{e}', v_2')$ in A with $\mathsf{e}' = \mathsf{e}$. This allows to write, when convenient, (v_1, e, v_2) as e.

Finally, given a direct graph representing a BPMN model, the concept of *path* is defined. A path $p \in G$ is a non-empty ordered sequence of arrows in A where the target vertex of an edge is also the source of the next edge in the sequence, for instance $p = (v_a, \mathsf{e}_1, v_f), (v_f, \mathsf{e}_2, v_b), (v_b, \mathsf{e}_3, v_r)$. A path that ends in its starting vertex is called a *cycle*. Given a path p of the form $(v_0, \mathsf{e}_0, v_1), \ldots, (v_{k-1}, \mathsf{e}_{k-1}, v_k)$, notations $\mathsf{first}(p)$ and $\mathsf{last}(p)$ indicate the starting edge e_0 and the ending edge e_{k-1} of p, respectively.

2.2.2 Running Example

Throughout this thesis, a running example consisting of two collaborations is used to help the reader in getting familiar with the BPMN modeling activity. This section presents both the business scenario, and the collaboration diagrams. More in detail, a larger collaboration diagram is presented with an almost complete set of BPMN elements. It is then used in Chapters 3 and 4 to introduce the operational semantics both formally and through animation. Then, a second collaboration with a smaller core set of BPMN elements is introduced. It appears in Chapter 5 as case study for the 3D animation.

The proposed collaboration diagrams depict procedures that come from the same scenario. It regards the submission of a paper to a peer reviewed journal, until its publication. The scenario involves different participants that act in one or both of the collaboration diagrams. They are:

- **Contact Author**, who contributes to the realization of the paper and performs the submission to the journal. Of course, she/he acts on behalf of the other authors;
- **Journal Handling Editor**, the editor of the journal, whose responsibility regards the assignment of the submitted paper to the reviewers, and the subsequent managing of the decision procedure;
- **Reviewer**, a person with knowledge in some of the journal topics who performs the reviewing activity. Since more than one reviewer takes part in this, this role is modeled as a process instance of a multi-instance pool;

- **Journal Publisher**, who takes charge of the publication of the accepted papers. She/he is also responsible for the copyright duties and the relative fee.

Paper Reviewing Collaboration. Figure 2.13 presents the first collaboration diagram, that regards the reviewing procedure. It concerns the submission of a paper from a selected author, i.e., the *Contact Author*, to a scientific journal and its management by the editor for the reviewing procedure. It involves three reviewers to judge the paper. Of course, the management of multiple papers submitted to the journal requires to enact the collaboration for each paper.

The paper reviewing collaboration starts once the *Contact Author* initiates its process contained in the homonymous pool, the top one in Figure 2.13. This is reflected by the start event *Paper Submission*, this enacts the following activity concerning the preparation of a draft of the paper, based on the information retrieved from the *Paper Data* data input. Then, the author writes the paper and in parallel, if necessary, gets the paper format and applies it. Then, once the manuscript is finalized, it is sent to the *Journal Editor* through the *Submit Manuscript* send task.

This triggers the message start event contained in the pool *Journal Handling Editor* and consequently enacts the inner process. He/she assigns the paper to three reviewers via the multi-instance sequential send task *Assign Paper* which loop cardinality is set to 3 according to the number of involved reviewers for each paper. The receiving of the *Review Request* message, hence of the paper to judge, starts an instance of the *Reviewer* process. Each instance prepares a review that contains the reviewer's name, a score, and a text. Access to a shared repository of research articles is available for each of the reviewers, it is depicted via the data store *Papers Repository.* When the review is ready, the reviewer sends it back to the editor that waits for receiving all three reviews.

After all reviews have been received, the editor stores them into the data object collection *Reviews* and combines the judgments looking at the scores in the reviews. According to the result of the *Reviews Evaluation* task, the editor prepares the acceptance/rejection letter (stored in the *Letter* data object) or, if the paper requires further discussion, postpones the decision. The decision behavior is rendered via a data-based XOR-Split gateway which relies on the review scores. Discussion interactions are here abstracted and always result in an accept or reject decision. Finally, feedback regarding the evaluations is sent to the reviewers, taking care of delivering each message to the right reviewer. Thus, depending on the review result, an acceptance or rejection notification is sent to the *Contact Author.* To finalize their processes, the reviewers can get the feedback once received through a message from the editor, or check it online repeatedly. In case of a delay in the

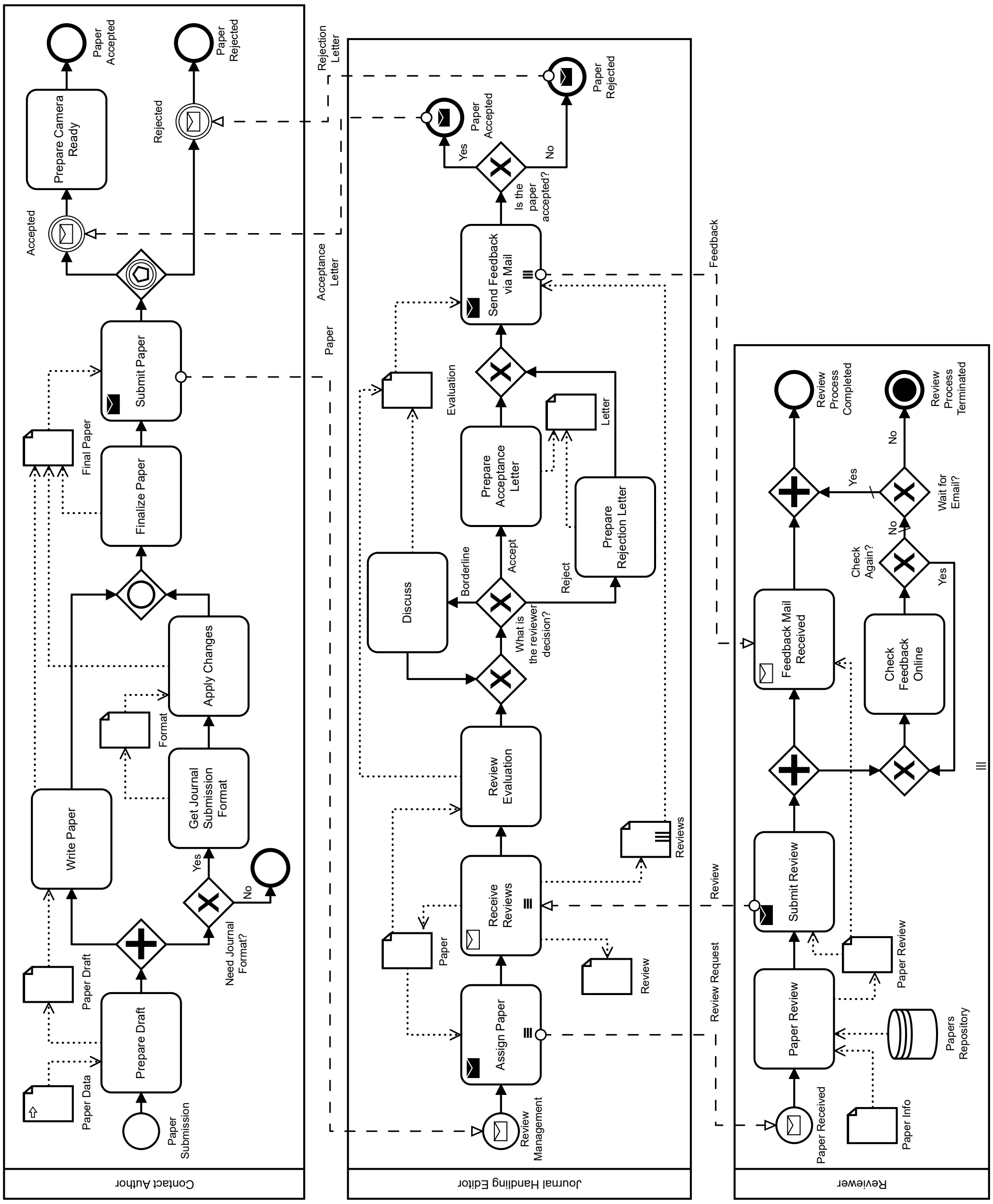


Figure 2.13: Paper reviewing collaboration model.

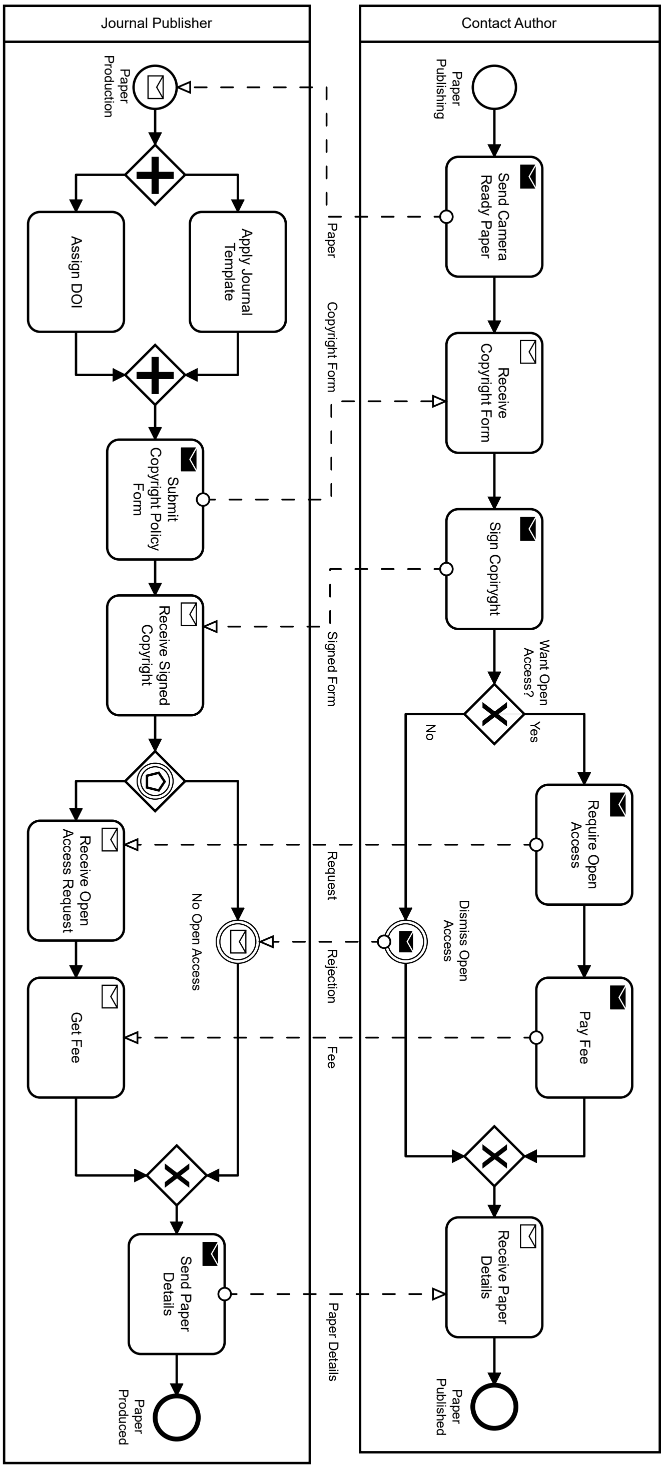


Figure 2.14: Paper publication collaboration model.

receiving of the e-mail, a reviewer can choose to abort its entire process by means of the terminate end event *Review Process Terminated.*

Paper Publication Collaboration. The collaboration presented so far ends with the acceptance or the rejection of the paper. The one in Figure 2.14 shows what happens next in case the paper has been accepted. Indeed, the *Paper Publication Collaboration* specifies the interactions that take place between the *Contact Author* and the *Journal Publisher*, in order to handle the production of the accepted paper.

The participants in this collaboration are the *Contact Author*, the same person who interacted with the editor, and the *Journal Publisher.* The collaboration starts once the author sends the revised paper to the publisher, activating its process. The publisher receives the paper and then, at the same time, applies to it a template to have a style compliant with the one of the journal, and assigns to it a Digital Object Identifier (DOI). Once finished, the publisher sends a copyright form to the author that, in its turn, fills, signs, and returns it. Subsequently, the *Contact Author* can choose to require an open access policy for the manuscript and pay the corresponding fee, otherwise he/she can refuse this possibility, and go on with the procedure. Depending on this choice, the editor will receive the money or not. Finally, he/she sends to the author the published paper with the publication details.

2.2.3 BPMN XML Representation

To foster interchangeability and the spreading of BPMN, the OMG permits to share diagrams in a standard manner. The OMG defines unique XML based notation, namely *.bpmn*, to exchange BPMN 2.0 diagrams between tools, companies, etc.

This format describes collaboration and process diagrams in a tree structured way, bringing all the information required for reproducing the elements composing the diagram, and their disposition. Indeed, each BPMN element can be mapped to an XML fragment containing semantic and visual information. The first part of the fragment depicts the semantic information (e.g., the element id, the connected sequence flows), while the second part spatially locates the element in the diagram.

More in general, the XML representation of the BPMN elements collects all the *attributes* regarding it, also those that are not visible graphically in the diagram. For instance, Listing 2.1 provides the corresponding XML fragment of the multi-instance send task *Assign Paper* in Figure 2.13. Within the fragment it is possible to identify: the element name and id (line 1), the incoming and the outgoing sequence flows (lines 2-3), the referenced data object (lines 4-8), and the multi-instance characteristics (lines 8-10).

The remaining lines belong to the extension element tag, which is used to include additional information with respect to the standard BPMN XML schema. We make use of these lines to express additional attributes that will be introduced by Chapter 3.

```
<bpmn:sendTask id="Assign_Paper" name="Assign Paper">
    <bpmn:incoming>SequenceFlow_0b3rh0j</bpmn:incoming>
    <bpmn:outgoing>SequenceFlow_0e7hug5</bpmn:outgoing>
    <bpmn:dataInputAssociation
        id="DataInputAssociation_0pz7bl9">
    <bpmn:sourceRef>DataObjectReference_15eg2ll</bpmn:sourceRef>
        <bpmn:targetRef>Property_0wufvcu</bpmn:targetRef>
    </bpmn:dataInputAssociation>
    <bpmn:multiInstanceLoopCharacteristics isSequential="true">
        <bpmn:loopCardinality
            xsi:type="bpmn:tFormalExpression">3</bpmn:loopCardinality>
    </bpmn:multiInstanceLoopCharacteristics>
</bpmn:sendTask>
<bpmndi:BPMNShape id="SendTask_00bygr0_di"
    bpmnElement="Assign_Paper">
    <dc:Bounds x="277" y="541" width="100" height="80" />
</bpmndi:BPMNShape>
```

Listing 2.1: Assign Paper XML fragment.

2.3 Operational Semantics

Providing a formal semantics of informal languages is an essential step to clearly define the requirements a system has to satisfy. In this regard, process algebras are mathematically rigorous languages with well defined semantics that permit describing and verifying properties of concurrent communicating systems [51].

The basic component of a process algebra is its syntax, that defines what well-formed terms are. Specifically, it is the combination of operators and more elementary terms. Many different approaches (operational, denotational, algebraic) can be used for describing the meaning of processes, that is to provide a semantics of the considered operators. However, the operational approach has become the reference one. By relying on the so called Structural Operational Semantics (SOS), an operational semantics models a process as a labelled transition system (LTS), that consists of a set of states, a set of transition labels and a transition relation. The states of the transition system are just process algebra terms while the labels of the transitions between states represent the actions or the interactions that are possible from a given state and the state that is reached after the action is performed by means of visible and invisible actions. Formally, an LTS [40] is defined as follows.

Definition 1 (Labeled Transition System). *A labeled state transition system is a tuple (S, Σ, δ) such that*

- *S is a finite set of states,*
- *Σ is a finite alphabet (i.e. a finite, non-empty set of symbols which refer to actions that a system can perform),*
- *$\delta \subset S \times \Sigma \times S$ is a state transition relation.*

Inference systems are used to associate LTSs to process terms and are defined as follows.

- *Inference Systems* are a set of inference rule of the form

  $$\frac{p_1, \dots, p_n}{q}$$

  where $p_1, \dots, p_n$ are the premises and q is the conclusion. Each rule is interpreted as an implication: if all premises are satisfied then also the conclusion is inferred.

- *Axioms* are rulel without premises and they are written as

  $$\frac{-}{q} \quad \text{or} \quad q$$

- *Transition Rules* represent transitions between states. In the case of operational semantics the premises and the conclusions will be triples of the form $P \xrightarrow{\ell} Q$ and thus the rules for each operator *op* of the process algebras will be like the one below, where $\{i_1, \dots, i_m\} \subseteq \{1, \dots, n\}$ and $E_i' = E_i$ when $i \notin \{i_1, \dots, i_m\}$.

  $$\frac{E_{i1} \xrightarrow{\ell_1} E_{i1}' \dots E_{im} \xrightarrow{\ell_m} E_{im}'}{op(E_1, \dots, E_n) \xrightarrow{\ell} C[E_1', \dots, E_n']}$$

  In the rule the targets term $C[\,]$ indicates the new context in which the new sub terms will be operating after the reduction. Sometimes, these rules are enriched with *side conditions* that determine their applicability. Therefore, transition rules, given a term representing a state of the system, permit to determine the enabled actions and the corresponding reachable states, thus defining an LTS.

- *Basic Actions* represent the atomic (uninterruptible) step of a computation that is performed by a system to move from one state to the next. Actions represent various activities of concurrent systems, like sending or receiving a message, synchronizing with other processes etc. In process algebras two main types of atomic actions are considered, namely *visible* (or external) actions and *invisible* (or internal actions), the latter referred by the Greek letter τ.

PART II

BPMN 2.0 FORMAL FRAMEWORK

CHAPTER 3

FORMALIZATION OF BPMN COLLABORATIONS

This chapter formalizes the semantics of BPMN collaborations. It focuses on those BPMN elements, informally presented in the previous chapter, that are strictly needed to deal with multiple instantiations of collaborations, namely multi-instance pools, message exchanges, multi-instance tasks (both in sequence and in parallel), data objects, data collections and data stores. Additionally, in order to define meaningful collaborations, some core BPMN elements (e.g., gateways and events) are considered. The rest of the chapter discusses the main issues related to the non trivial semantics of BPMN. To conclude, a comparison between the approach showed in this thesis and the literature is provided.

3.1 Multiple Instances, Messages and Data in a Nutshell

To deal with multiple instances in BPMN collaboration models, it is necessary to take into account the data flow. Indeed, the dynamic creation of process *instances* can be triggered by the arrival of *messages*, which contain data. Within a process instance, data can be accessed from *data objects, data collections and data stores* (here, and in the following, the term *data elements* is used to refer to all of them together), and drives the instance execution. Values of data elements can be used to fill the content of outgoing messages and, vice versa, the content of incoming messages can be stored in data elements. Below the interplay between such concepts is made clear . To this aim, Figure 3.1 depicts a colored version of the running example introduced in Section 2.2.2, where the elements composing respectively the *control*, *data*, and *message* flows have been colored respectively in blue, orange and green.

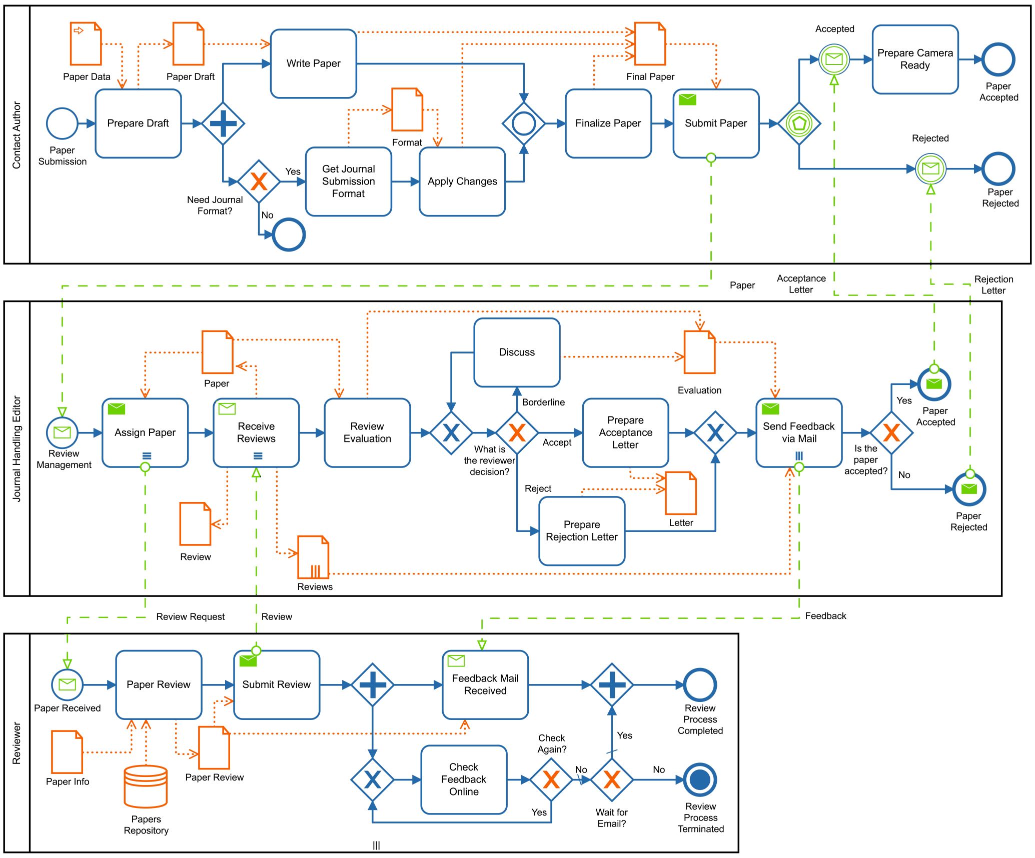


Figure 3.1: Control, data, and message flows in the running example.

Indeed, this collaboration model combines the activities of three participants. Each of them follows the behavior of its internal process which execution is driven by data, and finally, data is shared with the other processes by using messages. In this scenario, data support is crucial to precisely render the message exchanges between participants, especially because multiple instances of the *Reviewer* process are created. In fact, messages coming into this pool might start a new process instance, or be routed to existing instances already underway. Messages and process instances must contain enough information to determine, when a message arrives at a pool, if a new process instance is needed or, if not, which existing instance will handle it. To this aim, BPMN makes use of the concept of *correlation*: it is up to each single message to provide the information that permits to associate the message with the appropriate (possibly new) instance. This is achieved by embedding values, called *correlation data*, in the content of the message itself. As reported in the standard,

> *"Correlation is used to associate a particular Message [...] between two particular Process instances. BPMN allows using existing Message data for correlation purposes [...] rather than requiring the introduction of technical correlation data"* [53, Sec. 8.3.2].

In particular, this formalization relies on pattern-matching to enable the correlation of exchanged messages. Considering the running example, every time the *Journal Editor* sends a *Feedback* of the review to a *Reviewer*, the message must contain information (in this case the name of the reviewer) to be correlated to the correct process instance of the *Reviewer* multi-instance pool. In this way, the *Feedback* notification is properly delivered to the *Reviewer* instance.

According to the BPMN standard, data elements do not have any direct effect on the sequence flow or message flow of processes, since tokens do not flow along data associations [53, p. 221]. However, this statement is questionable. Indeed, on the one hand, the information stored in data elements can be used to drive the execution of process instances, as they can be referred in the conditional expressions of OR/XOR split gateways to take decisions about which branch should be taken. On the other hand, data elements can be connected in input to tasks. In particular, the standard states that

> *"the Data Objects as inputs into the Tasks act as an additional constraint for the performance of those Tasks. The performers [...] cannot start the Task without the appropriate input"* [53, p. 183].

In both cases, a data element has an implicit indirect effect on the execution, since it can drive the decision taken by a OR/XOR split gateway or acts as a guard condition on a task. In the running example, for instance, according to the value of the *Evaluation* data object, the conditional expression *What is the Reviewers Decision?* is evaluated and a branch of the XOR split gateway is chosen. As another example, the task *Submit Paper* in the *Contact Author* pool can be executed only if the field *finalized* of the *Final Paper* data object has been valorized to true.

Concerning the content of data elements, the standard left underspecified its structure, in order to keep the notation independent from the kind of data structure required from time to time. The presented formalism considers a generic record structure, assuming that a data object/store is just a list of fields, characterized by a name and the corresponding value. Instead, data collections are thought of as special data objects consisting of lists of elements that, in their own turn, are structured as lists of fields. Figure 3.2 reports the structure of the data elements used in the running example. Messages are structured as tuples of values; the latter can be manipulated and inserted into data element fields via assignments performed by tasks.

Guards, assignments, and structure of data elements and messages are

PaperData {journal, authors, format} PaperDraft {title, authors, format}
Format {style} FinalPaper {title, authors, format, text, finalized}

Paper {title, authors, body, score} Review {reviewer, title, body, score}
Reviews {reviewer, title, body, score}
Evaluation {title, decision} Letter {subject, evaluation}

PaperInfo {title, body} PaperReview {title, body, score, decision, myName}
PapersRepository {papers}

Figure 3.2: Structures of data elements of the running example.

not explicitly reported in the graphical representation of the BPMN model, but are defined as attributes of the involved BPMN elements.

3.2 Textual Notation of BPMN Collaborations

To simplify the formal treatment of the semantics, the formalism resorts to a textual representation of BPMN models, which is more manageable for writing operational rules than the graphical notation. Notice that this work does not propose an alternative modeling notation, but it just defines a Backus-Naur Form (BNF) syntax of BPMN model structures.

Figure 3.3 reports the BNF syntax defining the textual notation of BPMN collaboration models. This syntax only describes the structure of models. Notably, even if this syntax would allow to write collaborations that cannot be expressed in BPMN, here are considered only those terms of the syntax that can be derived from BPMN models.

In the proposed grammar, the non-terminal symbols C, P, T, T_{em}, N and A represent *Collaboration Structures*, *Process Structures*, *Task Structures*, *Task Execution Modalities*, *Non-Atomic Execution Modalities*, and *Data Assignments*, respectively. The terminal symbols, denoted by the sans serif font, are the typical elements of a BPMN model, i.e., pools, events, tasks and gateways. The syntax of these elements is based on the following disjoint sets:

- the set $\mathbb{P}$ of *pool names* (ranged over by p, p', $\ldots$);
- the set $\mathbb{E}$ of *sequence edges* (ranged over by e, e', e_i, $\ldots$) with $E \in 2^{\mathbb{E}}$ ranging over *sets of edges*;
- the set $\mathbb{M}$ of *message names* (ranged over by m, m', m_i, $\ldots$);
- the set $\mathbb{T}$ of *task names* (ranged over by t, t', $\ldots$);

- the set $\mathbb{C}$ of *counter names* (ranged over by c, c', ...);
- the set $\mathbb{D}$ of *data object names* (ranged over by do, do', ...);
- the set $\mathbb{D}_{\mathbb{S}}$ of *data store names* (ranged over by ds, ds', ...);
- the set $\mathbb{D}_{\mathbb{F}}$ of *data field names* (ranged over by f, f', ...);
- the set $\mathbb{F}$ of *data fields* (ranged over by $\mathsf{do.f}$, $\mathsf{ds.f}'$, ...); and
- the set $\mathbb{V}$ of *values* (ranged over by v, v', ...).

The syntax also uses a set $\mathbb{EXP}$ of *expressions* (ranged over by exp, exp', ...), whose precise syntax is deliberately not specified; it just assumes that expressions contain, at least, values v, data object fields $\mathsf{do.f}$ and data store fields $\mathsf{ds.f}$. Notation $\tilde{\cdot}$ denotes tuples; e.g., $\widetilde{\mathsf{exp}}$ stands for a tuple of expressions $\langle \mathsf{exp}_1, \ldots, \mathsf{exp}_n \rangle$.

$$
\begin{array}{lcl}
C & ::= & \mathsf{pool}(\mathsf{p}, P) \quad | \quad \mathsf{miPool}(\mathsf{p}, P, \mathsf{max}) \quad | \quad C \parallel C \\
P & ::= & \mathsf{start}(\mathsf{e}, \mathsf{e}') \quad | \quad \mathsf{startRcv}(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}) \quad | \quad \mathsf{end}(\mathsf{e}) \quad | \quad \mathsf{endSnd}(\mathsf{e}, \mathsf{m} : \widetilde{\mathsf{exp}}) \\
 & | & \mathsf{terminate}(\mathsf{e}) \quad | \quad \mathsf{interRcv}(\mathsf{e}, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}') \quad | \quad \mathsf{interSnd}(\mathsf{e}, \mathsf{m} : \widetilde{\mathsf{exp}}, \mathsf{e}') \\
 & | & \mathsf{andSplit}(\mathsf{e}, E) \quad | \quad \mathsf{xorSplit}(\mathsf{e}, G) \quad | \quad \mathsf{orSplit}(\mathsf{e}, G) \\
 & | & \mathsf{andJoin}(E, \mathsf{e}) \quad | \quad \mathsf{xorJoin}(E, \mathsf{e}) \quad | \quad \mathsf{orJoin}(E, \mathsf{e}) \\
 & | & \mathsf{eventBased}(\mathsf{e}, (\mathsf{m}_1 : \tilde{\mathsf{t}}_1, \mathsf{e}_1), \ldots, (\mathsf{m}_h : \tilde{\mathsf{t}}_h, \mathsf{e}_h)) \\
 & | & T \quad | \quad \mathsf{mipTask}(\mathsf{e}, \mathsf{exp}, T, \mathsf{c}, \mathsf{exp}', \mathsf{e}') \quad | \quad \mathsf{misTask}(\mathsf{e}, \mathsf{exp}, T, \mathsf{c}, \mathsf{exp}', \mathsf{e}') \\
 & | & P \parallel P \\
T & ::= & \mathsf{task}(\mathsf{e}, \mathsf{t}, T_{em}, \mathsf{exp}, A, \mathsf{e}') \quad | \quad \mathsf{taskRcv}(\mathsf{e}, \mathsf{t}, T_{em}, \mathsf{exp}, A, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}') \\
 & | & \mathsf{taskSnd}(\mathsf{e}, \mathsf{t}, T_{em}, \mathsf{exp}, A, \mathsf{m} : \widetilde{\mathsf{exp}}, \mathsf{e}') \\
T_{em} & ::= & \mathsf{a} \quad | \quad N \\
N & ::= & \mathsf{na\_c} \quad | \quad \mathsf{na\_nc} \\
A & ::= & \epsilon \quad | \quad \mathsf{do.f} := \mathsf{exp} \quad | \quad \mathsf{ds.f} := \mathsf{exp} \quad | \quad \mathsf{get}(\mathsf{do}) \quad | \quad \mathsf{push}(\mathsf{do}) \quad | \quad A,\, A
\end{array}
$$

Figure 3.3: BNF syntax of BPMN collaboration structures.

Intuitively, a BPMN collaboration model is rendered in the presented syntax as a collection of (single-instance and multi-instance) pools, each one specifying a process. Formally, a collaboration C is a composition, by means of the operator $\parallel$, of pools either of the form $\mathsf{pool}(\mathsf{p}, P)$ (for single-instance pools) or $\mathsf{miPool}(\mathsf{p}, P, \mathsf{max})$ (for multi-instance pools), where p is the name that uniquely identifies the pool, P is the enclosed process, and max is the maximum number of instances that can be activated in case of a multi-instance pool. Similarly, a process P is a composition of process elements by means of the operator $\|$.

The correspondence between the graphical notation of BPMN and the textual representation used here is straightforward, except for the terms $\mathsf{mipTask}$ and $\mathsf{misTask}$ where mip and mis stand for *multiple-instance parallel* and *multiple-instance sequential*, respectively. This correspondence is exemplified by means of the running example in Figure 3.4, where for the sake of readability the definition of those guard expressions that simply check the initialization of data fields are omitted (only exp_{101} is kept as an example).

For a more detailed account of the one-to-one correspondence the interested reader can refer to Tables 3.1 to 3.3. In the textual representation there is some information (content of messages, receiving templates, data element 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 characterization of the model. This information is explicitly reported in the textual representation as it is needed to properly define the execution semantics of the collaboration models. Moreover, to support a compositional approach, in the textual notation each sequence/message edge in the graphical notation is split in two parts: the part outgoing from the source element and the part incoming into the target element; the two parts are correlated by the unique edge name.

No direct syntactic representation of *data elements*, i.e., data objects, data collections and data stores is provided. The evolution of their states during the model execution is a semantic concern (described later in this chapter). Thus, syntactically, only the connections between data elements and the other process elements are relevant. They are rendered by references within *expressions*, used to check when a task is ready to start (graphically, the task has an incoming data association from the data element), to update the values stored in a data field (graphically, the task has outgoing data association to the data element), and to drive the decision of a XOR split gateway. The BPMN standard is quite loose in specifying what is the actual structure of data elements. A generic record structure for data objects and data stores is assumed to exist, so that a data object/store is just a list of fields, characterized by a name and the corresponding value. Specifically, the field named f of the data object named do (resp. the data store named ds) is accessed via the usual notation $\mathsf{do.f}$ (resp. $\mathsf{ds.f}$). A data collection instead is a special data object consisting of a list of elements that, in their own turn, are structured as lists of fields. The head element of a data collection do can be retrieved by means of $\mathsf{get(do)}$; as effect of the execution of this action, the fields of the retrieved element can be accessed as usual by means of $\mathsf{do.f}$. To

Overall paper reviewing collaboration scenario:

$$C = \text{pool}(\text{p}_{ca}, P_{ca}) \parallel \text{pool}(\text{p}_{jhe}, P_{jhe}) \parallel \text{miPool}(\text{p}_r, P_r, 3)$$

Contact Author process:

$$\begin{aligned}
P_{ca} =\ & \text{start}(\text{e}_{101}, \text{e}_{102}) \parallel \text{task}(\text{e}_{102}, \text{Prepare Draft}, \text{a}, \text{exp}_{101}, \text{A}_{101}, \text{e}_{103}) \parallel \text{andSplit}(\text{e}_{103}, \{(\text{e}_{104}, \text{e}_{105}\}) \parallel \\
& \text{task}(\text{e}_{104}, \text{Write Paper}, \text{a}, \text{exp}_{102}, \text{A}_{102}, \text{e}_{106}) \parallel \\
& \text{xorSplit}(\text{e}_{105}, \{(\text{e}_{107}, \text{PaperDraft.format} = \text{null}), (\text{e}_{108}, \text{PaperDraft.format} \neq \text{null})\}) \parallel \text{end}(\text{e}_{108}) \parallel \\
& \text{task}(\text{e}_{107}, \text{Get Journal Submission Format}, \text{a}, \text{true}, \text{A}_{103}, \text{e}_{109}) \parallel \\
& \text{task}(\text{e}_{109}, \text{Apply Changes}, \text{a}, \text{true}, \text{A}_{104}, \text{e}_{110}) \parallel \text{orJoin}(\{\text{e}_{106}, \text{e}_{110}\}, \text{e}_{111}) \parallel \\
& \text{task}(\text{e}_{111}, \text{Finalise Paper}, \text{a}, \text{true}, \text{A}_{105}, \text{e}_{112}) \parallel \\
& \text{taskSnd}(\text{e}_{112}, \text{Submit Paper}, \text{a}, \text{exp}_{103}, \epsilon, \text{Paper}:\tilde{\text{exp}}_{104}, \text{e}_{113}) \parallel \\
& \text{eventBased}(\text{e}_{113}, (\text{Acceptance Letter}:\langle\text{Final.paper}\rangle, \text{e}_{114}), (\text{Rejection Letter}:\langle\text{Final.paper}\rangle, \text{e}_{115})) \parallel \\
& \text{task}(\text{e}_{114}, \text{Prepare Camera Ready}, \text{a}, \text{true}, \epsilon, \text{e}_{116}) \parallel \text{end}(\text{e}_{116}) \parallel \text{end}(\text{e}_{117})
\end{aligned}$$

$$\begin{aligned}
\text{exp}_{101} =\ & \text{PaperData.journal} \neq \text{null} \ \& \ \text{PaperData.authors} \neq \text{null} \\
A_{101} =\ & \text{PaperDraft.title} := \text{'APaperTitle'},\ \text{PaperDraft.authors} := \text{PaperData.authors}, \\
& \text{PaperDraft.format} := \text{PaperData.format} \\
A_{102} =\ & \text{FinalPaper.title} := \text{PaperDraft.title},\ \text{FinalPaper.authors} := \text{PaperDraft.authors}, \\
& \text{Manuscript.text} := \text{'Lorem ipsum'},\ \text{FinalPaper.format} := \text{PaperDraft.format} \\
A_{103} =\ & \text{Format.style} := \text{'style'} \\
A_{104} =\ & \text{FinalPaper.format} := \text{Format.style} \\
A_{105} =\ & \text{FinalPaper.text} := \text{'Lorem ipsum dolor sit.'},\ \text{FinalPaper.finalised} := \text{true} \\
\text{exp}_{103} =\ & \text{FinalPaper.finalized} \\
\tilde{\text{exp}}_{104} =\ & \langle\text{FinalPaper.title}, \text{FinalPaper.authors}, \text{FinalPaper.text}\rangle
\end{aligned}$$

Journal Handling Editor process:

$$\begin{aligned}
P_{pc} =\ & \text{startRcv}(\text{Paper}:\tilde{\text{t}}_{201}, \text{e}_{202}) \parallel \\
& \text{misTask}(\text{e}_{202}, 3, \text{taskSnd}(\text{e}'_{202}, \text{Assign Paper}, \text{a}, \text{exp}_{201}, \epsilon, \text{Review Request}:\tilde{\text{exp}}_{201}, \text{e}'_{203}), \text{c}_{201}, \text{false}, \text{e}_{203}) \parallel \\
& \text{misTask}(\text{e}_{203}, 3, \text{taskRcv}(\text{e}'_{203}, \text{Receive Reviews}, \text{a}, \text{true}, \text{A}_{201}, \text{Review}:\tilde{\text{t}}_{202}, \text{e}'_{204}), \text{c}_{202}, \text{false}, \text{e}_{204}) \parallel \\
& \text{task}(\text{e}_{204}, \text{Review Evaluation}, \text{a}, \text{true}, \text{A}_{202}, \text{e}_{205}) \parallel \text{xorJoin}(\{\text{e}_{205}, \text{e}_{210}\}, \text{e}_{206}) \parallel \\
& \text{xorSplit}(\text{e}_{206}, \{(\text{e}_{207}, \text{Evaluation.decision} = 0), (\text{e}_{208}, \text{Evaluation.decision} > 0), (\text{e}_{209}, \text{Evaluation.decision} < 0)\}) \parallel \\
& \text{task}(\text{e}_{207}, \text{Discussion}, \text{a}, \text{true}, \text{A}_{203}, \text{e}_{210}) \parallel \text{task}(\text{e}_{208}, \text{Prepare Acceptance Letter}, \text{a}, \text{true}, \text{A}_{204}, \text{e}_{211}) \parallel \\
& \text{task}(\text{e}_{209}, \text{Prepare Rejection Letter}, \text{a}, \text{true}, \text{A}_{205}, \text{e}_{212}) \parallel \text{xorJoin}(\{\text{e}_{211}, \text{e}_{212}\}, \text{e}_{213}) \parallel \\
& \text{misTask}(\text{e}_{213}, 3, \text{taskSnd}(\text{e}'_{213}, \text{Send Feedback Mail}, \text{a}, \text{exp}_{202}, \text{A}_{206}, \text{Feedback}:\tilde{\text{exp}}_{202}, \text{e}'_{214}), \text{c}_{203}, \text{false}, \text{e}_{214}) \parallel \\
& \text{xorSplit}(\text{e}_{214}, \{(\text{e}_{215}, \text{Letter}_\text{e}\text{valuation} = \text{'Accepted'}), (\text{e}_{216}, \text{Letter}_\text{e}\text{valuation} = \text{'Rejected'})\}) \parallel \\
& \text{endSnd}(\text{e}_{215}, \text{Acceptance Letter}:\langle\text{Paper.title}\rangle) \parallel \text{endSnd}(\text{e}_{216}, \text{Rejection Letter}:\langle\text{Paper.title}\rangle)
\end{aligned}$$

$$\begin{aligned}
\tilde{\text{t}}_{201} =\ & \langle ?\text{Paper.title}, ?\text{Paper.authors}, ?\text{Paper.body}\rangle \\
\tilde{\text{exp}}_{201} =\ & \langle\text{Paper.title}, \text{Paper.body}\rangle \\
A_{201} =\ & \text{Reviews.reviewer} := \text{Review.reviewer},\ \text{Reviews.score} := \text{Review.score}, \text{Reviews.body} := \text{Review.body}, \\
& \text{Reviews.title} := \text{Review.title}, \text{push(Reviews)}, \text{Paper.score} := \text{Paper.score} + \text{Reviews.score} \\
\tilde{\text{t}}_{202} =\ & \langle ?\text{Review.reviewer}, \text{Paper.title}, ?\text{Paper.score}, ?\text{Paper.body}\rangle \\
A_{202} =\ & \text{Evaluation.title} := \text{'EvaluationofthePaper'},\ \text{Evaluation.decision} := \text{Paper.score} \\
A_{203} =\ & \text{Evaluation.decision} := \text{reevaluate(Reviews)} \\
A_{204} =\ & \text{Letter.subject} := \text{'AcceptanceLetter'},\ \text{Letter.evaluation} := \text{'Accepted'} \\
A_{205} =\ & \text{Letter.subject} := \text{'RejectionLetter}, \text{Letter.evaluation} := \text{'Rejected'} \\
A_{206} =\ & \text{get(Reviews)} \\
\tilde{\text{exp}}_{202} =\ & \langle\text{Reviews.title}, \text{Evaluation.decision}\rangle
\end{aligned}$$

Reviewer process:

$$\begin{aligned}
P_r =\ & \text{startRcv}(\text{Review Request}:\tilde{\text{t}}_{301}, \text{e}_{302}) \parallel \text{task}(\text{e}_{302}, \text{Paper Review}, \text{a}, \text{exp}_{301}, \text{A}_{301}, \text{e}_{303}) \parallel \\
& \text{taskSnd}(\text{e}_{303}, \text{Submit Review}, \text{a}, \text{exp}_{302}, \text{A}_{302}, \text{Review Request}:\tilde{\text{exp}}_{301}, \text{e}_{304}) \parallel \text{andSplit}(\text{e}_{304}, \{\text{e}_{305}, \text{e}_{306}\}) \parallel \\
& \text{taskRcv}(\text{e}_{305}, \text{Receive Feedback Mail}, \text{a}, \text{true}, \epsilon, \text{Feedback}:\tilde{\text{t}}_{302}, \text{e}_{307}) \parallel \text{xorJoin}(\{\text{e}_{306}, \text{e}_{311}\}, \text{e}_{308}) \parallel \\
& \text{task}(\text{e}_{308}, \text{Check Feedback Online}, \text{a}, \text{true}, \epsilon, \text{e}_{309}) \parallel \text{xorSplit}(\text{e}_{309}, \{(\text{e}_{310}, \text{isEnough}())(\text{e}_{311}, \text{isNotEnough}())\}) \parallel \\
& \text{andJoin}(\{\text{e}_{307}, \text{e}_{313}\}, \text{e}_{312}) \parallel \text{xorSplit}(\text{e}_{310}, \{(\text{e}_{313}, \text{waitForMail}()), (\text{e}_{314}, \text{dontWait}())\}) \parallel \\
& \text{end}(\text{e}_{312}) \parallel \text{terminate}(\text{e}_{314})
\end{aligned}$$

$$\begin{aligned}
\tilde{\text{t}}_{301} =\ & \langle ?\text{PaperInfo.title}, ?\text{PaperInfo.body}\rangle \\
A_{301} =\ & \text{PaperReview.title} := \text{PaperInfo.title},\ \text{PaperReview.score} := \text{judge}(), \\
& \text{PaperReview.body} := \text{'Review'} \\
A_{302} =\ & \text{PaperReview.myName} := \text{myName}() \\
\tilde{\text{exp}}_{301} =\ & \langle\text{PaperReview.myName}, \text{PaperReview.title}, \text{PaperReview.score}, \text{PaperReview.body}\rangle \\
\tilde{\text{t}}_{302} =\ & \langle\text{PaperReview.myName}, ?\text{PaperReview.title}, ?\text{PaperReview.decision}\rangle
\end{aligned}$$

Figure 3.4: Textual representation of the running example.

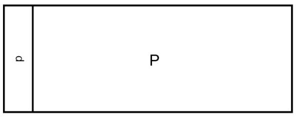
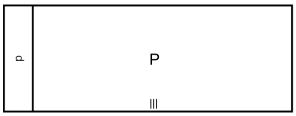
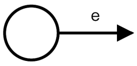
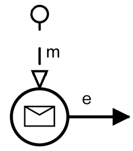
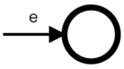
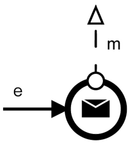
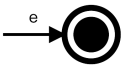
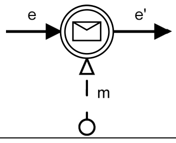
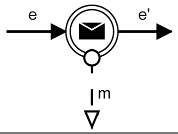
Graphical Notation	Textual Notation
	$\mathsf{pool}(\mathsf{p}, P)$
	$\mathsf{miPool}(\mathsf{p}, P, \mathsf{max})$
	$\mathsf{start}(\mathsf{e}', \mathsf{e})$
	$\mathsf{startRcv}(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e})$
	$\mathsf{end}(\mathsf{e})$
	$\mathsf{endSnd}(\mathsf{e}, \mathsf{m} : \mathsf{e\tilde{x}p})$
	$\mathsf{terminate}(\mathsf{e})$
	$\mathsf{interRcv}(\mathsf{e}, \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}')$
	$\mathsf{interSnd}(\mathsf{e}, \mathsf{m} : \mathsf{e\tilde{x}p}, \mathsf{e}')$

Table 3.1: Correspondence between graphical and textual notation: pools and events.

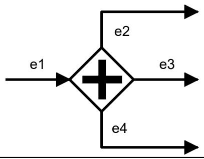
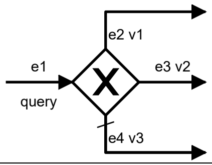
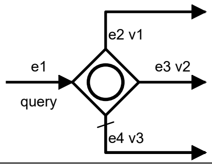
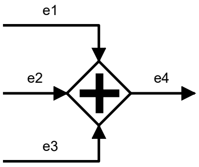
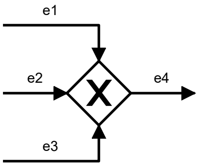
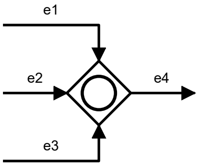
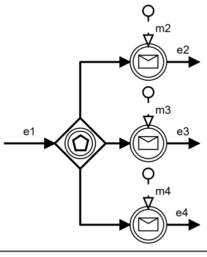
Graphical Notation	Textual Notation
	$\mathsf{andSplit}(\mathsf{e1}, \{\mathsf{e2}, \mathsf{e3}, \mathsf{e4}\})$
	$\mathsf{xorSplit}(\mathsf{e1}, \{(\mathsf{e2}, \mathsf{query} = \mathsf{v}_1), (\mathsf{e3}, \mathsf{query} = \mathsf{v}_2), (\mathsf{e4}, \mathsf{default})\})$
	$\mathsf{orSplit}(\mathsf{e1}, \{(\mathsf{e2}, \mathsf{query} = \mathsf{v}_1), (\mathsf{e3}, \mathsf{query} = \mathsf{v}_2), (\mathsf{e4}, \mathsf{default})\})$
	$\mathsf{andJoin}(\{\mathsf{e1}, \mathsf{e2}, \mathsf{e3}\}, \mathsf{e4})$
	$\mathsf{xorJoin}(\{\mathsf{e1}, \mathsf{e2}, \mathsf{e3}\}, \mathsf{e4})$
	$\mathsf{orJoin}(\{\mathsf{e1}, \mathsf{e2}, \mathsf{e3}\}, \mathsf{e4})$
	$\mathsf{eventBased}(\mathsf{e1}, (\mathsf{m2}\!:\!\tilde{\mathsf{t}}_2, \mathsf{e2}), (\mathsf{m3}\!:\!\tilde{\mathsf{t}}_3, \mathsf{e3}), (\mathsf{m4}\!:\!\tilde{\mathsf{t}}_4, \mathsf{e4}))$

Table 3.2: Correspondence between graphical and textual notation: gateways.

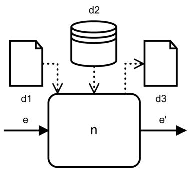
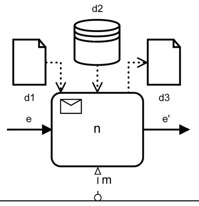
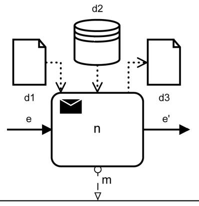
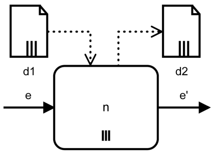
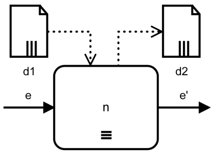
Graphical Notation	**Textual Notation**
	$\mathsf{task}(\mathsf{e}, \mathsf{t}, N, \mathsf{exp}(\mathsf{d}_1, \mathsf{d}_2),$ $(\mathsf{d}_3.\mathsf{f}_1 := \mathsf{exp}_1, \ldots, \mathsf{d}_3.\mathsf{f}_\mathsf{n} := \mathsf{exp}_\mathsf{n}), \mathsf{e}')$
	$\mathsf{taskRcv}(\mathsf{e}, \mathsf{t}, N, \mathsf{exp}(\mathsf{d}_1, \mathsf{d}_2),$ $(\mathsf{d}_3.\mathsf{f}_1 := \mathsf{exp}_1, \ldots, \mathsf{d}_3.\mathsf{f}_\mathsf{n} := \mathsf{exp}_\mathsf{n}), \mathsf{m} : \tilde{\mathsf{t}}, \mathsf{e}')$
	$\mathsf{taskSnd}(\mathsf{e}, \mathsf{t}, N, \mathsf{exp}(\mathsf{d}_1, \mathsf{d}_2),$ $(\mathsf{d}_3.\mathsf{f}_1 := \mathsf{exp}_1, \ldots, \mathsf{d}_3.\mathsf{f}_\mathsf{n} := \mathsf{exp}_\mathsf{n}), \mathsf{m} : \widetilde{\mathsf{exp}}, \mathsf{e}')$
	$\mathsf{mipTask}(\mathsf{e}, \mathsf{exp}, \mathsf{task}(\mathsf{e}'', \mathsf{t}, N, \mathsf{notEmpty}(\mathsf{d}_1),$ $(\mathsf{get}(\mathsf{d}_1), \mathsf{d}_2.\mathsf{f}_1 := \mathsf{exp}_1, \ldots, \mathsf{d}_2.\mathsf{f}_\mathsf{n} := \mathsf{exp}_\mathsf{n},$ $\mathsf{push}(\mathsf{d2})), \mathsf{e}'''), \mathsf{c}, \mathsf{exp}', \mathsf{e}')$
	$\mathsf{misTask}(\mathsf{e}, \mathsf{exp}, \mathsf{task}(\mathsf{e}'', \mathsf{t}, N, \mathsf{notEmpty}(\mathsf{d}_1),$ $(\mathsf{get}(\mathsf{d}_1), \mathsf{d}_2.\mathsf{f}_1 := \mathsf{exp}_1, \ldots, \mathsf{d}_2.\mathsf{f}_\mathsf{n} := \mathsf{exp}_\mathsf{n},$ $\mathsf{push}(\mathsf{d2})), \mathsf{e}'''), \mathsf{c}, \mathsf{exp}', \mathsf{e}')$

Table 3.3: Correspondence between graphical and textual notation: tasks.

add an element in a data collection do, first the fields of the new element are filled with values via assignments of the form $\mathsf{do.f} := \mathsf{exp}$, then the element with the filled fields is inserted in the tail of the data collection by means of $\mathsf{push}(\mathsf{do})$. It is worth noticing that in the presented semantics concrete values are associated to data object fields. The same applies to data stores and data collections. This perfectly fits with the purpose of *animating* the execution of collaboration models showing the evolution of the specified data.

Since data is explicitly considered, messages are characterized not only by labels, but also by the values that they may carry. Therefore, a sending action specifies a list of expressions whose evaluation will return a tuple of values to be sent, while a receiving action specifies a template to select matching messages and possibly assign values to data fields. Formally, a *message* is a pair $\mathsf{m} : \tilde{\mathsf{v}}$, where m is the (unique) message name (i.e., the label of the message edge) and $\tilde{\mathsf{v}}$ is a tuple of values representing the payload of the message. Sending actions have as argument a pair of the form $\mathsf{m}:\tilde{\mathsf{exp}}$. Receiving actions have as argument a pair of the form $\mathsf{m}:\tilde{\mathsf{t}}$, where $\tilde{\mathsf{t}}$ denotes a *template*, that is a sequence of expressions and formal fields used as pattern to select messages received by the pool. Formal fields are data object/store fields identified by the ?-tag (e.g., ?$\mathsf{do.f}$ or ?$\mathsf{ds.f}$) and are used to bind fields to values. Data elements are associated to a task by means of a conditional expression, which is a guard enabling the task execution, and a list of *assignments* A, each of which assigns the value of an expression to a data field or retrieves/inserts information in a data collection. When there is no data element as input to a task, the guard is simply true, while if there is no data element in output to a task the list of assignments is empty (ϵ).

The XOR split gateway as well as the OR split, specifies *guard conditions* in the 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. Notably, it is assumed that the set $\mathbb{EXP}$ of expressions includes the distinguished expression $\mathsf{default}$ referring to the *default sequence edge* outgoing from the gateway (it is assigned to at most one edge). When convenient, function G is considered as a set of pairs $(\mathsf{e}, \mathsf{exp})$.

Finally, the formalization supports the possibility of specifying the execution modality of tasks. This information is crucial when the data perspective and multi-instance tasks are taken into account. In case of a task with atomic execution (modality a), the evaluation of its enabling guard, the possible sending/receiving of a message, and the data object assignments, are performed atomically. This semantics fits well in many scenarios, like e.g., when a task acts on a data element representing a paper document managed by a human actor that cannot be accessed concurrently by other actors involved in the collaboration. However, there are also some situations where a non-atomic access is more suitable, e.g., when data elements represent shared digital documents. In the non-atomic case it is also important to indicate if the instances of a task can be executed concurrently (modality $\mathsf{na\_c}$) or not (modality $\mathsf{na\_nc}$). Actually, the BPMN standard is intentionally loose on these points, in order to allow the use of the modeling language in different contexts of use. To more effectively support designers, they can specify for each task the corresponding execution modality. This enables the identification of concurrency issues in those data accesses where they can actually arise and, at same time, it allows to ignore such issues when in the reality they cannot occur. The role of task execution modalities is particularly crucial in those cases where tasks act in parallel and access the same data elements. Parallel execution of tasks can produce in these cases different effects. This depends on the execution order of the internal steps of tasks, i.e. guard checks, message sending/receiving, and data element assignments. Considering, for instance, a simple scenario with two parallel tasks, each of which makes an assignment producing a violation of the guard of the other task. If the two tasks are atomic, the execution of one of them will be deadlocked, while in the non-atomic case such deadlock can be avoided if both tasks perform the guard checks before making the assignments. Concurrent and non-concurrent non-atomic modalities play an active role mainly when the involved tasks are multi-instance.

3.3 Semantics of BPMN Collaborations

The syntax presented so far represents the mere structure of processes and collaborations. To describe their semantics, the structural information is enriched with a notion of execution state, given by the marking of sequence edges with tokens [53, p. 27], the value of data elements, the status of tasks, and the exchanged messages. These stateful descriptions are respectively called process configurations and collaboration configurations, they produce local and global effects, respectively, on the process and collaboration execution. The operational semantics at collaboration level is defined by means of a *labelled transition system* (LTS), whose definition relies on an auxiliary LTS on the behavior of processes. Firstly, the process semantics is presented, and later the collaboration one.

3.3.1 Process Configuration

A *process configuration* has the form $\langle P, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_{ds}, \sigma_t, \sigma_c \rangle$, where:

- P is a process structure;
- $\sigma_e : \mathbb{E} \rightarrow \mathbb{N}$ is a *sequence edge state function* specifying, for each sequence edge, the current number of tokens marking it ($\mathbb{N}$ is indeed the set of natural numbers);
- $\sigma_{do} : \mathbb{F} \rightarrow \mathbb{V}$ is a *data object state function* assigning values (possibly null) to data object fields;
- $\sigma_{dc} : \mathbb{D} \rightarrow (\mathbb{F} \rightarrow \mathbb{V})^n$ is a *data collection state function* assigning to each data collection a tuple of data object state functions;
- $\sigma_{ds} : \mathbb{F} \rightarrow \mathbb{V}$ is a *data store state function* assigning values (possibly null) to data store fields; even if this state function has the same type as σ_{do}, the formalism uses two separate state functions because the information in data objects is treated differently from that in data stores, as this latter kind of data is permanent and shared among instances;

- $\sigma_t : \mathbb{T} \times \{a, s, r\} \rightarrow \mathbb{N}$ is a *task state function* used to keep track, for each non-atomic task, of the number of task instances in a given status, i.e., active (a), sending (s), and message received (r); the status of a task depends on its evolution (depicted in Figure 3.5), where the inactive status formally corresponds to haveing zero instances for all other statuses;

- $\sigma_c : \mathbb{C} \rightarrow \mathbb{N}$ is a *counter state function* used to keep track, for each multi-instance task, of the number of times that the task still has to be executed.

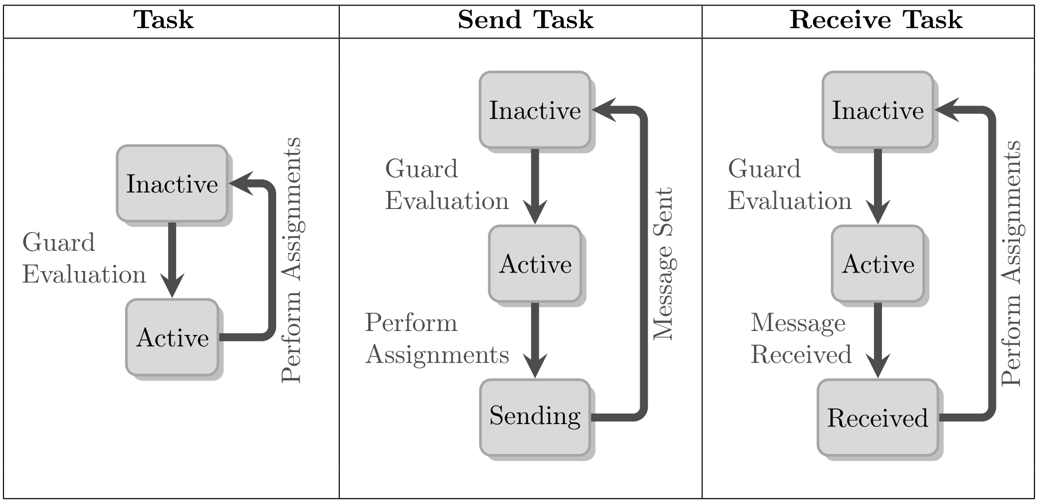


Figure 3.5: Task status evolution.

For the sake of presentation, in the following notation σ_d is used to denote in a compact way the triple $(\sigma_{do}, \sigma_{dc}, \sigma_{ds})$ representing the state of all data elements. Thus, a process configuration can be written for example as $\langle P, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle$. With σ_e^0 (resp. $\sigma_d^0, \sigma_t^0, \sigma_c^0$) is denoted the edge (resp. data element, task and counter) state where all edges are unmarked (resp. all data object/store fields are set to null, all data collections are empty, all tasks are inactive, and all counters are set to 0). Formally, $\sigma_e^0(\mathsf{e}) = 0\ \forall \mathsf{e} \in \mathbb{E}$, $\sigma_{do}^0(\mathsf{do.f}) = \mathsf{null}\ \ \forall\ \mathsf{do.f} \in \mathbb{F}$, $\sigma_{dc}^0(\mathsf{do}) = \epsilon\ \forall \mathsf{do} \in \mathbb{D}$, $\sigma_{ds}^0(\mathsf{ds.f}) = \mathsf{null}\ \ \forall\ \mathsf{ds.f} \in \mathbb{F}$, $\sigma_t^0(\mathsf{t}, a) = \sigma_t^0(\mathsf{t}, s) = \sigma_t^0(\mathsf{t}, r) = 0\ \forall \mathsf{t} \in \mathbb{T}$, and $\sigma_c^0(\mathsf{c}) = 0\ \forall \mathsf{c} \in \mathbb{C}$. The state obtained by updating in σ_e the number of tokens of the edge e to n, written as $\sigma_e \cdot [\mathsf{e} \mapsto \mathsf{n}]$, is defined as follows: $(\sigma_e \cdot [\mathsf{e} \mapsto \mathsf{n}])(\mathsf{e}')$ returns n if $\mathsf{e}' = \mathsf{e}$, otherwise it returns $\sigma_e(\mathsf{e}')$. The updates of states σ_{do}, σ_{dc}, σ_{ds}, σ_t, and σ_c are defined similarly.

Auxiliary Functions and Relations. To simplify the definition of the operational rules, some auxiliary functions and relations are defined to update states of process configurations and to act on the model topology.

- Function $inc(\sigma_e, \mathsf{e})$ increments by one the number of tokens marking the edge e in the state σ_e on σ_e.

  $$inc(\sigma_e, \mathsf{e}) = \sigma_e \cdot [\mathsf{e} \mapsto \sigma_e(\mathsf{e}) + 1]$$

  It extends in a natural way to sets E of edges. Specifically, inc is inductively defined as follows:

  $$\begin{array}{l} inc(\sigma_e, \emptyset) = \sigma_e \\ inc(\sigma_e, \{\mathsf{e}\} \cup E) = inc(inc(\sigma_e, \mathsf{e}), E) \end{array}$$

- Function $dec(\sigma_e, \mathsf{e})$ on σ_e decrements by one the number of tokens marking the edge e in the state σ_e on σ_e.

  $$dec(\sigma_e, \mathsf{e}) = \sigma_e \cdot [\mathsf{e} \mapsto \sigma_e(\mathsf{e}) - 1]$$

  It extends in a natural way to sets E of edges. Specifically, dec is inductively defined as follows:

  $$\begin{array}{l} dec(\sigma_e, \emptyset) = \sigma_e \\ dec(\sigma_e, \{\mathsf{e}\} \cup E) = dec(dec(\sigma_e, \mathsf{e}), E) \end{array}$$

- Function $reset(\sigma_e, \mathsf{e})$ sets to zero the number of tokens marking the edge e in the state σ_e.

  $$reset(\sigma_e, \mathsf{e}) = \sigma_e \cdot [\mathsf{e} \mapsto 0]$$

  It extends in a natural way to sets of edges as follows:

  $$\begin{array}{l} reset(\sigma_e, \emptyset) = \sigma_e \\ reset(\sigma_e, \{\mathsf{e}\} \cup E) = reset(reset(\sigma_e, \mathsf{e}), E) \end{array}$$

  Moreover, function $reset(\sigma_e) = \sigma_e^0$ resets all edges in the state σ_e.

- Function $set(\sigma_e, \mathsf{e}, h)$ sets to h the tokens marking the edge e in the state σ_e

  $$set(\sigma_e, \mathsf{e}, h) = \sigma_e \cdot [\mathsf{e} \mapsto h]$$

- Relation $eval(\mathsf{exp}, \sigma_d, \mathsf{v})$ states that v is one of the possible values resulting from the evaluation of the expression exp on the data element state σ_d; this is a relation, because an expression may contain non-deterministic operators, and is not explicitly defined, since the syntax of expressions is deliberately not specified (the only assumption is that $eval(\mathsf{default}, \sigma_d, \mathsf{v})$ implies $\mathsf{v} = \mathit{false}$ for any σ_d).

  Relations $eval(\tilde{\mathsf{exp}}, \sigma_d, \tilde{\mathsf{v}})$ and $eval(\tilde{\mathsf{t}}, \sigma_d, \tilde{\mathsf{et}})$ evaluate tuples of expressions and templates, respectively.

- Relation $upd(\sigma_d, A, \sigma'_d)$ states that σ'_d is one of the possible states resulting from the update of σ_d with assignment A. It is inductively defined as follows, for any $\sigma_{do}, \sigma_{dc}, \sigma_{ds}$:

$$upd((\sigma_{do}, \sigma_{dc}, \sigma_{ds}), \epsilon, (\sigma_{do}, \sigma_{dc}, \sigma_{ds}))$$

$$upd((\sigma_{do}, \sigma_{dc}, \sigma_{ds}), \mathsf{do.f} := \mathsf{exp}, (\sigma_{do} \cdot [\mathsf{do.f} \mapsto \mathsf{v}], \sigma_{dc}, \sigma_{ds}))$$

with v such that $eval(\mathsf{exp}, (\sigma_{do}, \sigma_{dc}, \sigma_{ds}), \mathsf{v})$.

$$upd((\sigma_{do}, \sigma_{dc}, \sigma_{ds}), \mathsf{ds.f} := \mathsf{exp}, (\sigma_{do}, \sigma_{dc}, \sigma_{ds} \cdot [\mathsf{ds.f} \mapsto \mathsf{v}]))$$

with v such that $eval(\mathsf{exp}, (\sigma_{do}, \sigma_{dc}, \sigma_{ds}), \mathsf{v})$.

$$upd((\sigma_{do}, \sigma_{dc}, \sigma_{ds}), \mathsf{get(do)}, (\sigma_{do} \cdot \sigma^1_{do}, \sigma'_{dc}, \sigma_{ds}))$$

with $\sigma_{dc}(\mathsf{do}) = \langle \sigma^1_{do}, \sigma^2_{do}, ..., \sigma^n_{do} \rangle$ and σ'_{dc} such that $\sigma'_{dc}(\mathsf{do}) = \langle \sigma^2_{do}, ..., \sigma^n_{do} \rangle$ and $\sigma'_{dc}(\mathsf{do}') = \sigma_{dc}(\mathsf{do}')$ with $\mathsf{do} \neq \mathsf{do}'$, where $\sigma_{do} \cdot \sigma'_{do} = \sigma_{do} \cdot [\mathsf{do_1.f_1} \mapsto \sigma'_{do}(\mathsf{do_1.f_1}), ..., \mathsf{do_n.f_n} \mapsto \sigma'_{do}(\mathsf{do_n.f_n})], \forall \mathsf{do_i.f_i} \in \mathbb{F}$ such that $\sigma'_{do}(\mathsf{do_i.f_i}) \neq \mathsf{null}$.

$$upd((\sigma_{do}, \sigma_{dc}, \sigma_{ds}), \mathsf{push(do)}, (\sigma_{do}, \sigma'_{dc}, \sigma_{ds}))$$

with $\sigma_{dc}(\mathsf{do}) = \langle \sigma^1_{do}, \sigma^2_{do}, ..., \sigma^n_{do} \rangle$ and σ'_{dc} such that $\sigma'_{dc}(\mathsf{do}) = \langle \sigma^1_{do}, \sigma^2_{do}, ..., \sigma^n_{do}, \sigma'_{do} \rangle$ and $\sigma'_{dc}(\mathsf{do}') = \sigma_{dc}(\mathsf{do}')$ for $\mathsf{do}' \neq \mathsf{do}$, and σ'_{do} such that $\sigma'_{do}(\mathsf{do}''.\mathsf{f}) = \sigma_{do}(\mathsf{do}''.\mathsf{f})$ if $\mathsf{do}'' = \mathsf{do}$ and $\sigma'_{do}(\mathsf{do}''.\mathsf{f}) = \mathsf{null}$ if $\mathsf{do}'' \neq \mathsf{do}$.

$$upd((\sigma_{do}, \sigma_{dc}, \sigma_{ds}), (A_1, A_2), (\sigma''_{do}, \sigma''_{dc}, \sigma''_{ds}))$$

with $\sigma''_{do}, \sigma''_{dc}, \sigma''_{ds}$ such that $upd((\sigma'_{do}, \sigma'_{dc}, \sigma'_{ds}), A_2, (\sigma''_{do}, \sigma''_{dc}, \sigma''_{ds}))$ and $\sigma'_{do}, \sigma'_{dc}, \sigma'_{ds}$ such that $upd((\sigma_{do}, \sigma_{dc}, \sigma_{ds}), A_1, (\sigma'_{do}, \sigma'_{dc}, \sigma'_{ds}))$.

- Function $inc(\sigma_t, \mathsf{t}, \mathsf{S})$ increments by one the number of instances of task t in the status $\mathsf{S} \in \{a, s, r\}$ in the state σ_t.

$$inc(\sigma_t, \mathsf{t}, \mathsf{S}) = \sigma_t \cdot [(\mathsf{t}, \mathsf{S}) \mapsto \sigma_t(\mathsf{t}, \mathsf{S}) + 1]$$

- Function $dec(\sigma_t, \mathsf{t}, \mathsf{S})$ decrements by one the number of instances of task t in the status $\mathsf{S} \in \{a, s, r\}$ in the state σ_t.

$$dec(\sigma_t, \mathsf{t}, \mathsf{S}) = \sigma_t \cdot [(\mathsf{t}, \mathsf{S}) \mapsto \sigma_t(\mathsf{t}, \mathsf{S}) - 1]$$

- Function $reset(\sigma_t)$ sets to inactive the status of all tasks in the state σ_t.

$$reset(\sigma_t) = \sigma^0_t$$

- Function $\mathit{isInactive}(\sigma_t, \mathsf{t})$ returns *true* if the task t is in the inactive status in the state σ_t.

$$\mathit{isInactive}(\sigma_t, \mathsf{t}) = (\sigma_t(\mathsf{t}, a) = 0 \wedge \sigma_t(\mathsf{t}, s) = 0 \wedge \sigma_t(\mathsf{t}, r) = 0)$$

- Function $\mathit{set}(\sigma_c, \mathsf{c}, h)$ sets to h the value of the counter c in the state σ_c.

$$\mathit{set}(\sigma_c, \mathsf{c}, h) = \sigma_c \cdot [\mathsf{c} \mapsto h]$$

- Function $\mathit{dec}(\sigma_c, \mathsf{c})$ decrements by one the value of c in the state σ_e.

$$\mathit{dec}(\sigma_c, \mathsf{c}) = \sigma_c \cdot [\mathsf{c} \mapsto \sigma_e(\mathsf{c}) - 1]$$

- Function $\mathit{reset}(\sigma_c, \mathsf{c})$ resets the value of the counter c in the state σ_c.

$$\mathit{reset}(\sigma_c, \mathsf{c}) = \sigma_c \cdot [\mathsf{c} \mapsto 0]$$

The auxiliary LTS of the behavior of processes is a triple $\langle \mathcal{P}, \mathcal{L}, \rightarrow \rangle$ where: $\mathcal{P}$ is a set of process configurations; $\mathcal{L}$, ranged over by ℓ, is a set of *labels*; and $\rightarrow \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ is a *transition relation.* As simplification, $\langle P, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle \xrightarrow{\ell} \langle P, \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle$ indicates that $(\langle P, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle, \ell, \langle P, \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle) \in \rightarrow$, and says that 'the process in the configuration $\langle P, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle$ can do a transition labelled by ℓ and evolve to the process configuration $\langle P, \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle$'. The labels used by the process transition relation are generated by the following production rules:

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

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 after the message m is actually received. Label $\mathit{new}\, \mathsf{m}:\tilde{\mathsf{et}}$ denotes the occurrence 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 the occurrence of the termination event.

Moreover, in order to catch the model semantics some functions that work on its topology are required.

Function $\mathit{edges}(P)$ permits to get the set of all edges used in the process P, and function $\mathit{in}(T)$ (resp. $\mathit{out}(T)$) to get the edge incoming in (resp. outgoing from) the task T.

$\mathbb{P}_{\mathbb{H}}$ refers to the set of all paths in G and $\mathcal{P}_{\mathcal{H}} : \mathbb{E} \rightarrow 2^{\mathbb{P}_{\mathbb{H}}}$ to a function that, given as input an edge $\mathsf{e} \in \mathbb{E}$ returns the set of all paths ending in the OR Join uniquely identified by e and starting from all the possible vertices between the start event and the OR Join, which do not visit the considered OR Join. Notably, this function returns a finite set of paths, because cycles within paths are not repeated.

Finally, to properly formalize the OR Join semantics in presence of vicious circles (i.e., keeping the execution blocked), for each OR Join has to be detected the presence of OR Joins from which it depends. This is expressed as a Boolean predicate $noDep : \mathbb{E} \rightarrow \{true, false\}$, which taken as input an edge e identifying an OR Join, holds if no other OR Join mutually depends on e.

Notational simplifications. To improve the readability of the operational rules, the following simplifications reduce the notation of transitions. More specifically, unnecessary information is omitted: *(i)* the states σ_e, σ_d, σ_t, σ_c from the source configuration of transitions, since the same notation is used for them in all rules; *(ii)* the structure from the target configuration of transitions, since process execution only affects the current states and not the process structure; *(iii)* those states from the target configuration that are not affected by transitions. Thus, for example, a transition $\langle P, \sigma_e, \sigma_d, \sigma_t, \sigma_c \rangle \xrightarrow{\ell} \langle P, \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c \rangle$ will be written as $P \xrightarrow{\ell} \langle \sigma'_e \rangle$ when it simply affects the sequence edge state function.

3.3.2 Process Semantics

The operational rules defining the transition relation of the process semantics are given by the inference rules in Figures 3.6, 3.8, 3.9, and 3.10.

Now a brief comment on the rules in Figure 3.6 is given. Rule *P-Start* starts the execution of a process when it has been activated. To denote the enabled status of start events an incoming (spurious) edge, named *enabling edge*, is included in their syntactical definition. Thus, the process is activated when the enabling edge of a start event is marked. The effect of the rule is to increment the number of tokens in the edge outgoing from the start event and to decrease 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 and forces the termination of the process instance by resetting the marking of edges and the status of tasks. Rule *P-StartRcv* starts the execution of a process by producing a label denoting the creation of a new instance and containing the information for consuming a received message at the collaboration layer (see rule *C-CreateMi* in Figure 3.11). Rule

$\mathsf{start}(\mathsf{e},\mathsf{e}') \xrightarrow{\epsilon} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle$	$\sigma_e(\mathsf{e}) > 0$	$(P\text{-}Start)$
$\mathsf{end}(\mathsf{e}) \xrightarrow{\epsilon} \langle dec(\sigma_e,\mathsf{e})\rangle$	$\sigma_e(\mathsf{e}) > 0$	$(P\text{-}End)$
$\mathsf{terminate}(\mathsf{e}) \xrightarrow{kill} \langle reset(\sigma_e), reset(\sigma_t)\rangle$	$\sigma_e(\mathsf{e}) > 0$	$(P\text{-}Terminate)$
$\mathsf{startRcv}(\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}) \xrightarrow{new\,\mathsf{m}\,:\,\tilde{\mathsf{et}}} \langle inc(\sigma_e,\mathsf{e})\rangle$	$eval(\tilde{\mathsf{t}},\sigma_d,\tilde{\mathsf{et}})$	$(P\text{-}StartRcv)$
$\mathsf{endSnd}(\mathsf{e},\mathsf{m}:\tilde{\mathsf{exp}}) \xrightarrow{!\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle dec(\sigma_e,\mathsf{e})\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\tilde{\mathsf{exp}},\sigma_d,\tilde{\mathsf{v}})$	$(P\text{-}EndSnd)$
$\mathsf{interRcv}(\mathsf{e},\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}')$ $\xrightarrow{?\mathsf{m}\,:\,\tilde{\mathsf{et}},\epsilon} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\ \tilde{\mathsf{t}}\ ,\sigma_d,\ \tilde{\mathsf{et}}\ )$	$(P\text{-}InterRcv)$
$\mathsf{interSnd}(\mathsf{e},\mathsf{m}:\tilde{\mathsf{exp}},\mathsf{e}')$ $\xrightarrow{!\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\tilde{\mathsf{exp}},\sigma_d,\tilde{\mathsf{v}})$	$(P\text{-}InterSnd)$
$\mathsf{andSplit}(\mathsf{e},E) \xrightarrow{\epsilon} \langle inc(dec(\sigma_e,\mathsf{e}),E)\rangle$	$\sigma_e(\mathsf{e}) > 0$	$(P\text{-}AndSplit)$
$\mathsf{xorSplit}(\mathsf{e},\{(\mathsf{e}',\mathsf{exp})\}\cup G)$ $\xrightarrow{\epsilon} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\mathsf{exp},\sigma_d,true)$	$(P\text{-}XorSplit_1)$
$\mathsf{xorSplit}(\mathsf{e},\{(\mathsf{e}',\mathsf{default})\}\cup G)$ $\xrightarrow{\epsilon} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $\forall(\mathsf{e}_j,\mathsf{exp}_j)\in G\ .$ $eval(\mathsf{exp}_j,\sigma_d,false)$	$(P\text{-}XorSplit_2)$
$\mathsf{orSplit}(\mathsf{e},\{(\mathsf{e}_1,\mathsf{exp}_1),\ldots,(\mathsf{e}_h,\mathsf{exp}_h)\}\cup G) \xrightarrow{\epsilon}$ $\langle inc(dec(\sigma_e,\mathsf{e}),\{(\mathsf{e}_1,\mathsf{exp}_1),\ldots,(\mathsf{e}_h,\mathsf{exp}_h)\})\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $\forall j\ s.t.\ 1\leqslant j\leqslant h\ .$ $eval(\mathsf{exp}_j,\sigma_d,true)$	$(P\text{-}OrSplit)$
$\mathsf{orSplit}(\mathsf{e},\{(\mathsf{e}',\mathsf{default})\}\cup G)$ $\xrightarrow{\epsilon} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $\forall(\mathsf{e}_j,\mathsf{exp}_j)\in G\ .$ $eval(\mathsf{exp}_j,\sigma_d,false)$	$(P\text{-}OrSplit_2)$
$\mathsf{andJoin}(E,\mathsf{e}) \xrightarrow{\epsilon} \langle inc(dec(\sigma_e,E),\mathsf{e})\rangle$	$\forall\mathsf{e}'\in E\ .\ \sigma_e(\mathsf{e}') > 0$	$(P\text{-}AndJoin)$
$\mathsf{xorJoin}(\{\mathsf{e}\}\cup E,\mathsf{e}') \xrightarrow{\epsilon} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle$	$\sigma_e(\mathsf{e}) > 0$	$(P\text{-}XorJoin)$
$\mathsf{orJoin}(E_1\sqcup E_2,\mathsf{e}') \xrightarrow{\epsilon} \langle inc(dec(\sigma_e,E_1),\mathsf{e}')\rangle$	$\forall\mathsf{e}_i\in E_1\ .\ \sigma_e(\mathsf{e}_i) > 0\ \wedge$ $\forall\mathsf{e}_j\in E_2\ .\ \sigma_e(\mathsf{e}_j) = 0\ \wedge$ $E_1\neq\varnothing\ \wedge$ $\forall p_1\in\mathit{\Pi}.\exists\, p_2\in\mathit{\Pi}_{p_1}$	$(P\text{-}OrJoin)$
$\mathsf{eventBased}(\mathsf{e},(\mathsf{m}_1:\tilde{\mathsf{t}}_1,\mathsf{e}_1),\ldots,(\mathsf{m}_h:\tilde{\mathsf{t}}_h,\mathsf{e}_h))$ $\xrightarrow{?\mathsf{m}_j\,:\,\tilde{\mathsf{et}}_j,\epsilon} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}_j)\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge\ \ \exists\, 1\leqslant j\leqslant h.$ $eval(\ \tilde{\mathsf{t}_j}\ ,\sigma_d,\ \tilde{\mathsf{et}_j}\ )$	$(P\text{-}EventG)$

Figure 3.6: BPMN process semantics: events and gateways.

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 rule *C-DeliverMi* in Figure 3.11). Rules *P-InterRcv* and *P-InterSnd* are enabled when there is at least a token in their incoming edge and move it to their outgoing edge, while producing a receive or a send label, respectively. 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 the tokens 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 all guard expressions are evaluated to *false*; in this case the default edge is marked. Rule *P-OrSplit* is activated when there is a token in the incoming edge of an OR-Split gateway, which is then removed while a token is added in all the outgoing edges where the conditional expression is evaluated to *true*[1]. Rule *P-OrSplit$_2$* is applied when all guard expressions are evaluated to *false*; in this case the default edge is marked. 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.

The OR Join semantics is quite complex, Figure 3.7 distills its characteristics from a detailed reading of the BPMN specification (as a matter of terminology, Inclusive Gateway stands for OR Join, while Sequence Flow for sequence edge).

From the standard it is clear that the OR Join has a *non-local semantics* and its activation may depend on the marking evolution considering the whole diagram. More in detail, given an OR Join with a token in at least one of its incoming edges, it has to wait for a token that is in a path ending in an empty incoming edge of such OR Join that does not visit the OR Join itself. However, if this token is also in a path ending in a non-empty incoming edge, then the OR Join is activated and the execution can proceed.

[1] Notably, in the rule operator $\sqcup$ denotes the disjoint union of sets, i.e. $E_1 \sqcup E_2$ stands for $E_1 \cup E_2$ if $E_1 \cap E_2 = \emptyset$, it is undefined otherwise.

Rule *P-OrJoin* defines the semantics of the OR Join gateway. The operator $\sqcup$ is used to split the set of edges incoming in the OR Join into two disjoint sets, E_1 and E_2, such that one contains marked edges ($\forall \mathsf{e}' \in E_1.\sigma_e(\mathsf{e}') > 0$) and the other one contains unmarked edges ($\forall \mathsf{e}' \in E_2.\sigma_e(\mathsf{e}') = 0$).

The Inclusive Gateway is activated if:

- At least one incoming Sequence Flow has at least one token and
- For every directed path formed by sequence flow that:
    - (i) starts with a Sequence Flow *f* of the diagram that has a token,
    - (ii) ends with an incoming Sequence Flow of the inclusive gateway that has no token,
    - (iii) does not visit the Inclusive Gateway.
- There is also a directed path formed by Sequence Flow that:
    - (iv) starts with *f*,
    - (v) ends with an incoming Sequence Flow of the inclusive gateway that has a token,
    - (vi) does not visit the Inclusive Gateway.

Figure 3.7: OR Join semantics according to the OMG standard BPMN 2.0.

In describing the rule, the BPMN 2.0 specification is exploited to make clear the correspondence. *"The Inclusive Gateway is activated if"* the conditions for the rule applications are satisfied. Thus, the requirement *"At least one incoming Sequence Flow has at least one token"* is represented by condition $E_1 \neq \emptyset$. The second requirement *"For every directed path formed by Sequence Flow that (i)... (ii)... (iii)... There is also a directed path formed by Sequence Flow that (iv)... (v)... (vi)"* is represented by the condition $\forall p_1 \in \mathit{\Pi}.\exists\, p_2 \in \mathit{\Pi}_{p_1}$, where $\mathit{\Pi}$ is the set of paths satisfying (i), (ii) and (iii), while the sets $\mathit{\Pi}_p$, one for each path p in $\mathit{\Pi}$, contain paths satisfying (iv), (v) and (vi). Formally, they are defined as $\mathit{\Pi} = \{p \in \mathcal{P}_{\mathcal{H}}(\mathsf{e}) \mid \sigma_e(\mathsf{first}(p)) > 0 \,\wedge\, \mathsf{last}(p) \in E_2\}$ and $\mathit{\Pi}_p = \{p' \in \mathcal{P}_{\mathcal{H}}(\mathsf{e}) \mid \mathsf{first}(p') = \mathsf{first}(p) \,\wedge\, \mathsf{last}(p') \in E_1\}$. In particular, a path p in $\mathit{\Pi}$ is such that: "*(i) starts with a Sequence Flow f of the diagram that has a token*" ($\sigma_e(\mathsf{first}(p)) > 0$), "*(ii) ends with an incoming Sequence Flow of the inclusive gateway that has no token*" ($\mathsf{last}(p) \in E_2$), and "*(iii) does not visit the Inclusive Gateway*" (ensured by definition of $\mathcal{P}_{\mathcal{H}}$). Instead, given a path p in $\mathit{\Pi}$, a path p' in $\mathit{\Pi}_p$ is such that: "*(iv) starts with f*" ($\mathsf{first}(p') = \mathsf{first}(p)$, as f is the first edge of p), "*(v) ends with an incoming Sequence Flow of the inclusive gateway that has a token*" ($\mathsf{last}(p') \in E_1$), and "*(vi) does not visit the Inclusive Gateway*" (ensured again by definition of $\mathcal{P}_{\mathcal{H}}$).

The rules in Figure 3.8 deal with task execution, both atomic and non-atomic. Starting with the non-atomic tasks, rule *P-Task$_A$* deals with non-

$\mathsf{task}(\mathsf{e},\mathsf{t},\mathsf{a},\mathsf{exp},A,\mathsf{e}')$ $\xrightarrow{\epsilon} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}'),\sigma'_d\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\mathsf{exp},\sigma_d, true)\ \wedge$ $upd(\sigma_d, A, \sigma'_d)$	$(P\text{-}Task_A)$
$\mathsf{taskRcv}(\mathsf{e},\mathsf{t},\mathsf{a},\mathsf{exp},A,\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}')$ $\xrightarrow{?\mathsf{m}\,:\,\tilde{\mathsf{et}},A} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\mathsf{exp},\sigma_d, true)\ \wedge$ $eval(\,\tilde{\mathsf{t}}\,,\sigma_d,\,\tilde{\mathsf{et}}\,)$	$(P\text{-}TaskRcv_A)$
$\mathsf{taskSnd}(\mathsf{e},\mathsf{t},\mathsf{a},\mathsf{exp},A,\mathsf{m}:\widetilde{\mathsf{exp}},\mathsf{e}')$ $\xrightarrow{!\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}'),\sigma'_d\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\mathsf{exp}',\sigma_d, true)\ \wedge$ $upd(\sigma_d, A, \sigma'_d)\ \wedge$ $eval(\widetilde{\mathsf{exp}},\sigma_d,\tilde{\mathsf{v}})$	$(P\text{-}TaskSnd_A)$
$\mathsf{task}(\mathsf{e},\mathsf{t},N,\mathsf{exp},A,\mathsf{e}')$ $\xrightarrow{\epsilon} \langle dec(\sigma_e,\mathsf{e}), inc(\sigma_t,\mathsf{t},\mathsf{a})\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\mathsf{exp},\sigma_d, true)\ \wedge$ $N = \mathsf{na\_nc} \Rightarrow isInactive(\sigma_t,\mathsf{t})$	$(P\text{-}Task_{N1})$
$\mathsf{task}(\mathsf{e},\mathsf{t},N,\mathsf{exp},A,\mathsf{e}')$ $\xrightarrow{\epsilon} \langle inc(\sigma_e,\mathsf{e}'),\sigma'_d, dec(\sigma_t,\mathsf{t},\mathsf{a})\rangle$	$\sigma_t(\mathsf{t},\mathsf{a}) > 0\ \wedge$ $upd(\sigma_d, A, \sigma'_d)$	$(P\text{-}Task_{N2})$
$\mathsf{taskRcv}(\mathsf{e},\mathsf{t},N,\mathsf{exp},A,\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}')$ $\xrightarrow{\epsilon} \langle dec(\sigma_e,\mathsf{e}), inc(\sigma_t,\mathsf{t},\mathsf{a})\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\mathsf{exp},\sigma_d, true)\ \wedge$ $N = \mathsf{na\_nc} \Rightarrow isInactive(\sigma_t,\mathsf{t})$	$(P\text{-}TaskRcv_{N1})$
$\mathsf{taskRcv}(\mathsf{e},\mathsf{t},N,\mathsf{exp},A,\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}')$ $\xrightarrow{?\mathsf{m}\,:\,\tilde{\mathsf{et}},\epsilon} \langle inc(dec(\sigma_t,\mathsf{t},\mathsf{a}),\mathsf{t},\mathsf{r})\rangle$	$\sigma_t(\mathsf{t},\mathsf{a}) > 0\ \wedge$ $eval(\,\tilde{\mathsf{t}}\,,\sigma_d,\,\tilde{\mathsf{et}}\,)$	$(P\text{-}TaskRcv_{N2})$
$\mathsf{taskRcv}(\mathsf{e},\mathsf{t},N,\mathsf{exp},A,\mathsf{m}:\tilde{\mathsf{t}},\mathsf{e}')$ $\xrightarrow{\epsilon} \langle inc(\sigma_e,\mathsf{e}'),\sigma'_d, dec(\sigma_t,\mathsf{t},\mathsf{r})\rangle$	$\sigma_t(\mathsf{t},\mathsf{r}) > 0\ \wedge$ $upd(\sigma_d, A, \sigma'_d)$	$(P\text{-}TaskRcv_{N3})$
$\mathsf{taskSnd}(\mathsf{e},\mathsf{t},N,\mathsf{exp},A,\mathsf{m}:\widetilde{\mathsf{exp}},\mathsf{e}')$ $\xrightarrow{\epsilon} \langle dec(\sigma_e,\mathsf{e}), inc(\sigma_t,\mathsf{t},\mathsf{a})\rangle$	$\sigma_e(\mathsf{e}) > 0\ \wedge$ $eval(\mathsf{exp},\sigma_d, true)\ \wedge$ $N = \mathsf{na\_nc} \Rightarrow isInactive(\sigma_t,\mathsf{t})$	$(P\text{-}TaskSnd_{N1})$
$\mathsf{taskSnd}(\mathsf{e},\mathsf{t},N,\mathsf{exp},A,\mathsf{m}:\widetilde{\mathsf{exp}},\mathsf{e}')$ $\xrightarrow{\epsilon} \langle \sigma'_d, inc(dec(\sigma_t,\mathsf{t},\mathsf{a}),\mathsf{t},\mathsf{s})\rangle$	$\sigma_t(\mathsf{t},\mathsf{a}) > 0\ \ \wedge upd(\sigma_d, A, \sigma'_d)$	$(P\text{-}TaskSnd_{N2})$
$\mathsf{taskSnd}(\mathsf{e},\mathsf{t},N,\mathsf{exp},A,\mathsf{m}:\widetilde{\mathsf{exp}},\mathsf{e}')$ $\xrightarrow{!\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle inc(\sigma_e,\mathsf{e}'), dec(\sigma_t,\mathsf{t},\mathsf{s})\rangle$	$\sigma_t(\mathsf{t},\mathsf{s}) > 0\ \ \wedge eval(\widetilde{\mathsf{exp}},\sigma_d,\tilde{\mathsf{v}})$	$(P\text{-}TaskSnd_{N3})$

Figure 3.8: BPMN process semantics: tasks with atomic and non-atomic executions.

communicating tasks, possibly equipped with data objects. It is activated only when the guard expression exp is satisfied and there is a token in the incoming edge, which is then moved to the outgoing edge. The rule also updates the values of the data objects connected in output to the task by performing the assignments A. Rule *P-TaskRcv*$_A$ is similar, but it produces a label corresponding to the consumption of a message. In this case, however, the data updates are not executed, because they must be done only after the message is actually received; therefore, the assignments are passed by means of the label to the collaboration layer (see rule *C-ReceiveMi* in Figure 3.11). Rule *P-TaskSnd*$_A$ sends a message, updates the data object and moves the incoming token to the outgoing edge. The produced send label is used to deliver the message at the collaboration layer (see rule *C-DeliverMi* in Figure 3.11).

The remaining rules deal with tasks with non-atomic execution, with both concurrent and non-concurrent modalities. According to the evolution of the task status, shown in Figure 3.5, the execution of non-communicating tasks is split in two steps: task activation (rule *P-Task*$_{N1}$), dealing with the evaluation of the guard and consumption of the token in the incoming edge, and task completion (rule *P-Task*$_{N2}$), dealing with the execution of the assignments and the insertion of the token in the outgoing edge. Notably, in case of non-concurrent execution, the task activation is performed only if the task is in the inactive status (i.e., there are no active instances). Similarly, the execution of receiving/sending tasks is split in three steps: task activation, receiving/sending of the message while the task is running, and task completion. Again, non-concurrent tasks are activated only if they are in the inactive status.

The rules in Figure 3.9 deal with multi-instance tasks, both in parallel and in sequence. A parallel multi-instance task is activated when it is inactive (i.e., $\sigma_c(\mathsf{c}) = 0$) and has an incoming token. If the *loop cardinality* expression exp is evaluated to a natural number h greater than 0 (rule *P-MipTask*$_1$), this value is assigned to the task counter c, and h tokens are inserted in the incoming edge of the wrapped task T. Instead, if the loop cardinality is 0 (rule *P-MipTask*$_2$), no execution of T is performed and the incoming token is moved directly to the outgoing edge. When the multi-instance task is active, the wrapped task can be executed according to the task rules previously described (rule *P-MipTask*$_3$). Finally, the multi-instance task completes (rule *P-MipTask*$_4$) when either all task instances have completed their execution (i.e., the number of tokens in the outgoing edge of T is equal to the loop cardinality stored in the counter c) or the *completion condition* expression exp' is evaluated to *true*. Rules for sequential multi-instance task are similar, thus just the key differences are discussed. When the multi-instance task is activated, only one token is inserted in the incoming edge of the wrapped

$$\begin{array}{r}\mathsf{mipTask}(\mathsf{e},\mathsf{exp},T,\mathsf{c},\mathsf{exp}',\mathsf{e}')\\ \xrightarrow{\epsilon}\langle set(dec(\sigma_e,\mathsf{e}),in(T),h),set(\sigma_c,\mathsf{c},h)\rangle\end{array} \qquad \begin{array}{l}\sigma_e(\mathsf{e})>0\ \wedge\\ \sigma_c(\mathsf{c})=0\ \wedge\\ eval(\mathsf{exp},\sigma_d,h)\\ \quad with\ h>0\end{array} \qquad (P\text{-}MipTask_1)$$

$$\begin{array}{r}\mathsf{mipTask}(\mathsf{e},\mathsf{exp},T,\mathsf{c},\mathsf{exp}',\mathsf{e}')\\ \xrightarrow{\epsilon}\langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle\end{array} \qquad \begin{array}{l}\sigma_e(\mathsf{e})>0\ \wedge\\ \sigma_c(\mathsf{c})=0\ \wedge\\ eval(\mathsf{exp},\sigma_d,0)\end{array} \qquad (P\text{-}MipTask_2)$$

$$\frac{\langle T,\sigma_e,\sigma_d,\sigma_t,\sigma_c\rangle\xrightarrow{\ell}\langle\sigma'_e,\sigma'_d,\sigma'_t,\sigma'_c\rangle}{\mathsf{mipTask}(\mathsf{e},\mathsf{exp},T,\mathsf{c},\mathsf{exp}',\mathsf{e}')\xrightarrow{\ell}\langle\sigma'_e,\ \sigma'_d,\sigma'_t,\sigma'_c\rangle} \qquad (P\text{-}MipTask_3)$$

$$\begin{array}{r}\mathsf{mipTask}(\mathsf{e},\mathsf{exp},T,\mathsf{c},\mathsf{exp}',\mathsf{e}')\\ \xrightarrow{\epsilon}\langle inc(reset(\sigma_e,edges(T)),\mathsf{e}'),reset(\sigma_c,\mathsf{c})\rangle\end{array} \qquad \begin{array}{l}\sigma_e(out(T))=\sigma_c(\mathsf{c})\\ \vee\\ eval(\mathsf{exp}',\sigma_d,true)\end{array} \qquad (P\text{-}MipTask_4)$$

$$\begin{array}{r}\mathsf{misTask}(\mathsf{e},\mathsf{exp},T,\mathsf{c},\mathsf{exp}',\mathsf{e}')\\ \xrightarrow{\epsilon}\langle inc(dec(\sigma_e,\mathsf{e}),in(T)),set(\sigma_c,\mathsf{c},h)\rangle\end{array} \qquad \begin{array}{l}\sigma_e(\mathsf{e})>0\ \wedge\\ \sigma_c(\mathsf{c})=0\ \wedge\\ eval(\mathsf{exp},\sigma_d,h)\\ \quad with\ h>0\end{array} \qquad (P\text{-}MisTask_1)$$

$$\begin{array}{r}\mathsf{misTask}(\mathsf{e},\mathsf{exp},T,\mathsf{c},\mathsf{exp}',\mathsf{e}'),\\ \xrightarrow{\epsilon}\langle inc(dec(\sigma_e,\mathsf{e}),\mathsf{e}')\rangle\end{array} \qquad \begin{array}{l}\sigma_e(\mathsf{e})>0\ \wedge\\ \sigma_c(\mathsf{c})=0\ \wedge\\ eval(\mathsf{exp},\sigma_d,0)\end{array} \qquad (P\text{-}MisTask_2)$$

$$\frac{\langle T,\sigma_e,\sigma_d,\sigma_t,\sigma_c\rangle\xrightarrow{\ell}\langle\sigma'_e,\ \sigma'_d,\sigma'_t,\sigma'_c\rangle}{\mathsf{misTask}(\mathsf{e},\mathsf{exp},T,\mathsf{c},\mathsf{exp}',\mathsf{e}')\xrightarrow{\ell}\langle\sigma'_e,\ \sigma'_d,\sigma'_t,\sigma'_c\rangle} \qquad (P\text{-}MisTask_3)$$

$$\begin{array}{r}\mathsf{misTask}(\mathsf{e},\mathsf{exp},T,\mathsf{c},\mathsf{exp}',\mathsf{e}')\\ \xrightarrow{\epsilon}\langle inc(dec(\sigma_e,out(T)),in(T)),dec(\sigma_c,\mathsf{c})\rangle\end{array} \qquad \begin{array}{l}\sigma_c(\mathsf{c})>1\ \wedge\\ \sigma_e(out(T))=1\ \wedge\\ eval(\mathsf{exp}',\sigma_d,false)\end{array} \qquad (P\text{-}MisTask_4)$$

$$\begin{array}{r}\mathsf{misTask}(\mathsf{e},\mathsf{exp},T,\mathsf{c},\mathsf{exp}',\mathsf{e}')\\ \xrightarrow{\epsilon}\langle inc(dec(\sigma_e,out(T)),\mathsf{e}'),reset(\sigma_c,\mathsf{c})\rangle\end{array} \qquad \begin{array}{l}\sigma_e(out(T))=1\ \wedge\\ (\sigma_c(\mathsf{c})=1\ \vee\\ eval(\mathsf{exp}',\sigma_d,true))\end{array} \qquad (P\text{-}MisTask_5)$$

Figure 3.9: BPMN process semantics: parallel/sequential multi-instance tasks.

task (rule *P-MisTask$_1$*). Then, when the wrapped task produces a token in its outgoing edge, indicating its termination, if the multi-instance task has not completed its execution then the wrapped task is reactivated and the counter decreased (rule *P-MisTask$_4$*).

$$\frac{\langle P_1, \sigma_e, \sigma_d, \sigma_t, \sigma_c\rangle \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c\rangle}{P_1 \parallel P_2 \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c\rangle}\ (\textit{P-Int}_1) \qquad \frac{\langle P_2, \sigma_e, \sigma_d, \sigma_t, \sigma_c\rangle \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c\rangle}{P_1 \parallel P_2 \xrightarrow{\ell} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c\rangle}\ (\textit{P-Int}_2)$$

Figure 3.10: BPMN process semantics: interleaving.

The last group of rules, *P-Int$_1$* and *P-Int$_2$* in Figure 3.10, deal with interleaving of process elements in a standard way, so that if an element of a process evolves then the whole process evolves accordingly.

3.3.3 Collaboration Configuration

Now consider the semantics at collaboration level. A *collaboration configuration* has the form $\langle C, \sigma_i, \sigma_m, \sigma_{ds}\rangle$, where:

- C is a collaboration structure;
- $\sigma_i : \mathbb{P} \rightarrow 2^{\mathbb{S}_{\sigma_e} \times \mathbb{S}_{\sigma_{do}} \times \mathbb{S}_{\sigma_{dc}} \times \mathbb{S}_{\sigma_t} \times \mathbb{S}_{\sigma_c}}$ is an *instance state function* mapping each pool name to a multiset of instance states, ranged over by I and containing quintuples of the form $\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c\rangle$ (where $\mathbb{S}_\sigma$ is the set of states of type σ);
- $\sigma_m : \mathbb{M} \rightarrow 2^{\mathbb{V}^n}$ is a *message state function* that assigns to each message name m a multiset of value tuples representing the messages received along the message edge labelled by m;
- σ_{ds} is a *data store state function*, defined as for process configurations.

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

Auxiliary Functions. To simplify the definition of the operational rules, some auxiliary functions, presented in the following, permit to update states of collaboration configurations.

- Function $newI(\sigma_i, \mathsf{p}, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c)$ adds the new instance $\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c \rangle$ to the multiset of instances of pool p in the state σ_i.

  $$newI(\sigma_i, \mathsf{p}, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c) = \sigma_i \cdot [\mathsf{p} \mapsto \sigma_i(\mathsf{p}) + \{\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c \rangle\}]$$

- Function $updI(\sigma_i, \mathsf{p}, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c, I)$ replaces an existing instance of p in the state σ_i, leaving instances I unaltered.

  $$updI(\sigma_i, \mathsf{p}, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c, I) = \sigma_i \cdot [\mathsf{p} \mapsto \{\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c \rangle\} + I]$$

- Function $add(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}})$ adds the value tuple $\tilde{\mathsf{v}}$ for the message name m in the state σ_m.

  $$add(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}) = \sigma_m \cdot [\mathsf{m} \mapsto \sigma_m(\mathsf{m}) + \{\tilde{\mathsf{v}}\}]$$

- Function $rm(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}})$ removes the message with value tuple $\tilde{\mathsf{v}}$ for the message name m from the state σ_m.

  $$rm(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}) = \sigma_m \cdot [\mathsf{m} \mapsto \sigma_m(\mathsf{m}) - \{\tilde{\mathsf{v}}\}]$$

- Function $match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}})$ is a partial function performing *pattern-matching* on structured data (like in [57]), 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 patter-matching function *match* is inductively defined as follows:

  $$\begin{array}{l} match(\mathsf{v}, \mathsf{v}) = \epsilon \\ match(?\mathsf{do.f}, \mathsf{v}) = (\mathsf{do.f} := \mathsf{v}) \\ match(?\mathsf{ds.f}, \mathsf{v}) = (\mathsf{ds.f} := \mathsf{v}) \\ match((\mathsf{et}', \tilde{\mathsf{et}}), (\mathsf{v}', \tilde{\mathsf{v}})) = match(\mathsf{et}', \mathsf{v}'), match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) \end{array}$$

Returning to the running example, the scenario in its initial state is rendered as the collaboration configuration

$$\langle (\mathsf{pool}(\mathsf{p}_{ca}, P_{ca}) \parallel \mathsf{pool}(\mathsf{p}_{jhe}, P_{jhe}) \parallel \mathsf{miPool}(\mathsf{p}_r, P_r, 3)), \sigma_i, \sigma_m, \sigma_{ds} \rangle$$

where: $\sigma_i(\mathsf{p}_{ca}) = \{\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c \rangle\}$ with $\sigma_e = \sigma_e^0 \cdot [\mathsf{e}_{101} \mapsto 1]$ and $\sigma_{do} = \sigma_{do}^0 \cdot [\mathsf{PaperData.journal}, \mathsf{PaperData.authors}, \mathsf{PaperData.format} \mapsto$

$'Journal\ Name', 'Names\ and\ Surnames', \mathsf{null}]$; and $\sigma_i(\mathsf{p}_{jhe}) = \sigma_i(\mathsf{p}_r) = \emptyset$. Notice that the σ_{do} function of the p_{ca} instance is initialised with the content of the PaperData data input.

The labelled transition relation on collaboration configurations formalises the message exchange and the data update according to the process evolution. The LTS is a triple $\langle \mathcal{C}, \mathcal{L}_c, \rightarrow_c \rangle$ where: $\mathcal{C}$ is a set of collaboration configurations; $\mathcal{L}_c$, ranged over by l, is a set of labels; and $\rightarrow_c \subseteq \mathcal{C} \times \mathcal{L}_c \times \mathcal{C}$ is a transition relation. The same readability simplifications used for process configuration transitions has been considered again. The labels used by the collaboration transition relation are generated by the following grammar:

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

Notably, internal and sending labels coincide with the same labels at the process level, while the receiving labels here just keep track of the received message.

3.3.4 Collaboration Semantics

The operational rules defining the transition relation of the collaboration semantics are given in Figure 3.11. In the following we propos a discussion on the relevant points. 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 a 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. Anyway, the instance is created only if the maximum number of allowed instances is not exceeded. The next three rules allow a single pool, representing organization p, to evolve according to the evolution of one of its process instances. In particular, if the process instance performs an internal action (rule *C-InternalMi*) or a receiving/delivery action (rules *C-ReceiveMi* or *C-DeliverMi*), the pool performs the corresponding action at collaboration layer. As for instance creation, rule *C-ReceiveMi* can be applied only if there is at least one matching message. Recall indeed that at process level the receiving labels just indicate the willingness of a process instance to consume a received message, regardless of the actual presence of messages. The delivering of messages is based on the *correlation* mechanism: the correlation data are identified by the template fields that are not formal (i.e., those fields requiring specific matching values). Moreover, when a process performs a sending action, the message state function is updated in order to deliver the sent message to the receiving participant. Finally, rules *C-Int*$_1$ and *C-Int*$_2$ permit to interleave the execution of actions performed by pools

$$\frac{\begin{array}{c}\sigma_i(\mathsf{p}) = \varnothing \qquad \langle P, \sigma_e^0, \sigma_d^0, \sigma_t^0, \sigma_c^0\rangle \xrightarrow{\mathit{new}\,\mathsf{m}\,:\,\tilde{\mathsf{et}}} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c\rangle \\ \tilde{\mathsf{v}} \in \sigma_m(\mathsf{m}) \qquad \mathit{match}(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) = A \quad \mathit{upd}(\sigma'_d, A, \sigma''_{do}, \sigma''_{dc}, \sigma''_{ds})\end{array}}{\mathsf{pool}(\mathsf{p}, P) \xrightarrow{\mathit{new}\,\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle \mathit{newI}(\sigma_i, \mathsf{p}, \sigma'_e, \sigma''_{do}, \sigma''_{dc}, \sigma'_t, \sigma'_c), \mathit{rm}(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}), \sigma''_{ds}\rangle}\ (\textit{C-Create})$$

$$\frac{\begin{array}{c}|\sigma_i(\mathsf{p})| < \mathsf{max} \qquad \langle P, \sigma_e^0, \sigma_d^0, \sigma_t^0, \sigma_c^0\rangle \xrightarrow{\mathit{new}\,\mathsf{m}\,:\,\tilde{\mathsf{et}}} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c\rangle \\ \tilde{\mathsf{v}} \in \sigma_m(\mathsf{m}) \qquad \mathit{match}(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) = A \quad \mathit{upd}(\sigma'_d, A, \sigma''_{do}, \sigma''_{dc}, \sigma''_{ds})\end{array}}{\mathsf{miPool}(\mathsf{p}, P, \mathsf{max}) \xrightarrow{\mathit{new}\,\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle \mathit{newI}(\sigma_i, \mathsf{p}, \sigma'_e, \sigma''_{do}, \sigma''_{dc}, \sigma'_t, \sigma'_c), \mathit{rm}(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}), \sigma''_{ds}\rangle}\ (\textit{C-CreateMi})$$

$$\frac{\begin{array}{c}\sigma_i(\mathsf{p}) = \{\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c\rangle\} \\ \langle P, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_{ds}, \sigma_t, \sigma_c\rangle \xrightarrow{\tau} \langle \sigma'_e, \sigma'_{do}, \sigma'_{dc}, \sigma'_{ds}, \sigma'_t, \sigma'_c\rangle\end{array}}{\mathsf{pool}(\mathsf{p}, P) \xrightarrow{\tau} \langle \mathit{updI}(\sigma_i, \mathsf{p}, \sigma'_e, \sigma'_{do}, \sigma'_{dc}, \sigma'_t, \sigma'_c, \varnothing), \sigma'_{ds}\rangle}\ (\textit{C-Internal})$$

$$\frac{\begin{array}{c}\sigma_i(\mathsf{p}) = \{\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c\rangle\} + I \\ \langle P, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_{ds}, \sigma_t, \sigma_c\rangle \xrightarrow{\tau} \langle \sigma'_e, \sigma'_{do}, \sigma'_{dc}, \sigma'_{ds}, \sigma'_t, \sigma'_c\rangle\end{array}}{\mathsf{miPool}(\mathsf{p}, P, \mathsf{max}) \xrightarrow{\tau} \langle \mathit{updI}(\sigma_i, \mathsf{p}, \sigma'_e, \sigma'_{do}, \sigma'_{dc}, \sigma'_t, \sigma'_c, I), \sigma'_{ds}\rangle}\ (\textit{C-InternalMi})$$

$$\frac{\begin{array}{c}\sigma_i(\mathsf{p}) = \{\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c\rangle\} \\ \langle P, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_{ds}, \sigma_t, \sigma_c\rangle \xrightarrow{?\mathsf{m}\,:\,\tilde{\mathsf{et}}, A} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c\rangle \\ \tilde{\mathsf{v}} \in \sigma_m(\mathsf{m}) \qquad \mathit{match}(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) = A' \qquad \mathit{upd}(\sigma'_d, (A', A), \sigma''_{do}, \sigma''_{dc}, \sigma''_{ds})\end{array}}{\mathsf{pool}(\mathsf{p}, P) \xrightarrow{?\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle \mathit{updI}(\sigma_i, \mathsf{p}, \sigma'_e, \sigma''_{do}, \sigma''_{dc}, \sigma'_t, \sigma'_c, I), \mathit{rm}(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}), \sigma''_{ds}\rangle}\ (\textit{C-Receive})$$

$$\frac{\begin{array}{c}\sigma_i(\mathsf{p}) = \{\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c\rangle\} + I \\ \langle P, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_{ds}, \sigma_t, \sigma_c\rangle \xrightarrow{?\mathsf{m}\,:\,\tilde{\mathsf{et}}, A} \langle \sigma'_e, \sigma'_d, \sigma'_t, \sigma'_c\rangle \\ \tilde{\mathsf{v}} \in \delta(\mathsf{m}) \qquad \mathit{match}(\tilde{\mathsf{et}}, \tilde{\mathsf{v}}) = A' \qquad \mathit{upd}(\sigma'_d, (A', A), \sigma''_{do}, \sigma''_{dc}, \sigma''_{ds})\end{array}}{\mathsf{miPool}(\mathsf{p}, P, \mathsf{max}) \xrightarrow{?\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle \mathit{updI}(\sigma_i, \mathsf{p}, \sigma'_e, \sigma''_{do}, \sigma''_{dc}, \sigma'_t, \sigma'_c, I), \mathit{rm}(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}), \sigma''_{ds}\rangle}\ (\textit{C-ReceiveMi})$$

$$\frac{\begin{array}{c}\sigma_i(\mathsf{p}) = \{\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c\rangle\} \\ \langle P, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_{ds}, \sigma_t, \sigma_c\rangle \xrightarrow{!\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle \sigma'_e, \sigma'_{do}, \sigma'_{dc}, \sigma'_{ds}, \sigma'_t, \sigma'_c\rangle\end{array}}{\mathsf{pool}(\mathsf{p}, P) \xrightarrow{!\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle \mathit{updI}(\sigma_i, \mathsf{p}, \sigma'_e, \sigma'_{do}, \sigma'_{dc}, \sigma'_t, \sigma'_c, I), \mathit{add}(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}), \sigma'_{ds}\rangle}\ (\textit{C-Deliver})$$

$$\frac{\begin{array}{c}\sigma_i(\mathsf{p}) = \{\langle \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_t, \sigma_c\rangle\} + I \\ \langle P, \sigma_e, \sigma_{do}, \sigma_{dc}, \sigma_{ds}, \sigma_t, \sigma_c\rangle \xrightarrow{!\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle \sigma'_e, \sigma'_{do}, \sigma'_{dc}, \sigma'_{ds}, \sigma'_t, \sigma'_c\rangle\end{array}}{\mathsf{miPool}(\mathsf{p}, P, \mathsf{max}) \xrightarrow{!\mathsf{m}\,:\,\tilde{\mathsf{v}}} \langle \mathit{updI}(\sigma_i, \mathsf{p}, \sigma'_e, \sigma'_{do}, \sigma'_{dc}, \sigma'_t, \sigma'_c, I), \mathit{add}(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}), \sigma'_{ds}\rangle}\ (\textit{C-DeliverMi})$$

$$\frac{\langle C_1, \sigma_i, \sigma_m, \sigma_{ds}\rangle \xrightarrow{l} \langle \sigma'_i, \sigma'_m, \sigma'_{ds}\rangle}{C_1 \parallel C_2 \xrightarrow{l} \langle \sigma'_i, \sigma'_m, \sigma'_{ds}\rangle}\ (\textit{C-Int}_1) \qquad\qquad \frac{\langle C_2, \sigma_i, \sigma_m, \sigma_{ds}\rangle \xrightarrow{l} \langle \sigma'_i, \sigma'_m, \sigma'_{ds}\rangle}{C_1 \parallel C_2 \xrightarrow{l} \langle \sigma'_i, \sigma'_m, \sigma'_{ds}\rangle}\ (\textit{C-Int}_2)$$

Figure 3.11: BPMN collaboration semantics.

of the same collaboration, so that if a part of a larger collaboration evolves, then the whole collaboration evolves accordingly.

3.4 Lessons Learned

The BPMN standard has the flavor of a framework rather than of a concrete language, because some aspects are not covered by it, but left to the designer [79]. For example, the standard leaves unspecified the internal structure of data objects:

"*Data Object elements can optionally reference a DataState element [...] The definition of these states, e.g., possible values and any specific semantics are out of scope of this specification*" [53, p. 206].

The situation does not change if one refers to the internal structure of data stores and data collections. This gap left by the BPMN standard must be filled in order to concretely deal with data in the formalization. To this aim, this formalism considers a generic record structure for data elements. Similarly, the expression language operating on data is left unspecified by the standard. This is not an issue for the formalization, but the expression language has to be instantiated in the concrete implementation of the semantics, for instance in developing an animator or a simulator.

The BPMN standard also lacks a clear description of the task execution modality. This formalism contributes to fill this gap thanks to its capability to manage different modalities of task execution taking into account atomicity and concurrency. This feature is explained by means of the process model in Figure 3.12. It provides a minimal example whose execution consists in performing firstly *Task A*, then *Task B* and *Task C* in parallel, and when both complete, the process ends. All these tasks can access the data object *Data* composed by the three fields a, b, and c. Guards and assignments are specified as follows: (i) *Task A* has *true* as guard condition and performs the assignment $Data.a := 1$; (ii) *Task B* has $Data.a = 1$ as guard and performs $Data.b := 2$ and $Data.a := 0$ as assignments; and (iii) *Task C* has $Data.a = 1$ as guard and performs $Data.c := 5$ and $Data.a := 0$ as assignments. The execution of this process can produce different results depending on the execution modality setting of its tasks. In case all tasks are executed with the atomic modality, the assignment to the field a performed by the firstly executed task between *Task B* and *Task C* disables the other task (because it makes the task's guard become *false*). Therefore, regardless of the order of execution of the parallel tasks, the execution of the overall process never reaches the end. In the non-atomic case, instead, different process executions can take place, as the task execution is now split in two steps: *(i)* guard evaluation, and *(ii)* assignments execution. Depending on how these steps of the execution of *Task B* and *Task C* interleave, the process may end properly or not. Finally, in case the tasks could be activated more than once at the same time (e.g., in case they would be multi-instance tasks, or in case of an unsafe process), the overall behavior would be also affected by the setting of the concurrent execution modality, which may allow or not the interleaved execution among instances of the same task.

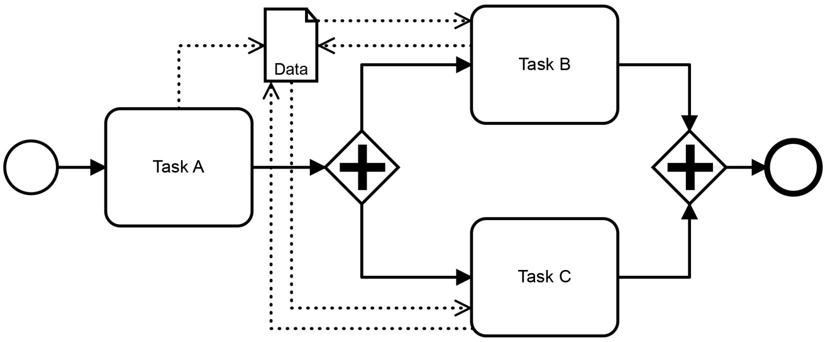
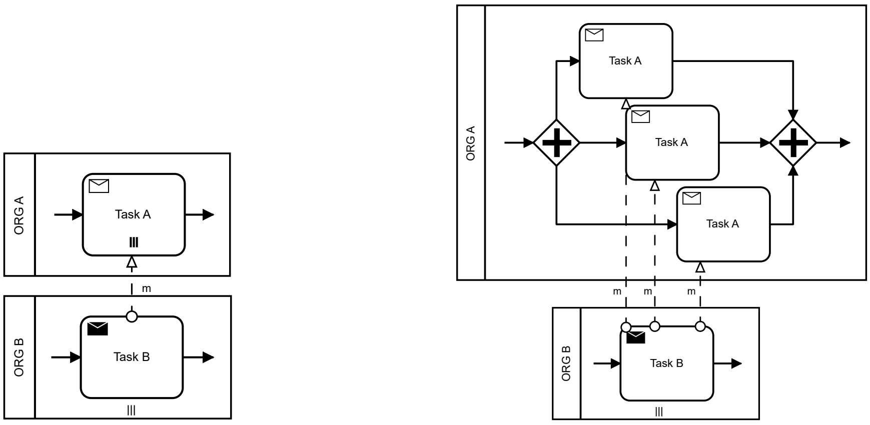


Figure 3.12: Atomic vs. non-atomic task execution.

In addition, the lack of formal semantics in the standard may lead to different interpretations of the tricky features of BPMN. This work aims at clarifying the interplay between multiple instances, messages and data objects. In particular, the standard provides an informal description of the mechanism used to correlate messages and process instances [53, p. 74], which has been formalized and implemented by following the solution adopted by the standard for executable business processes [52]. However, the BPMN standard does not provide any hint on how instances of the same process can communicate with each other. This may be particular useful in practical scenarios when instances of the same pool have to coordinate (in case, e.g., they need to achieve an agreement on a shared decision). To this aim, a data store serves to share persistent information among the instances.

Finally, even if from the semantic point of view it is common practice to consider multi-instance tasks as macros, the proposed work provides a direct characterization of their semantics. Concerning the sequential case, it is clear that the multi-instance task can simply be dealt with as a macro: it corresponds to a task enclosed within a sort of 'for' loop. Indeed, this was the solution I adopted in [17]. However, to keep track of the number of executed instances it is necessary to add to the model a further data object, to be used as a counter, for each multi-instance task. Moreover, to provide a complete specification of this BPMN element, a loop cardinality expression and a completion condition have to be considered. Even if formally sound, the use of this macro alters the original model, increasing its complexity, and hence is not practical in supporting tools. This is why we introduce a syntactic term for sequential multi-instance tasks with its own semantic



(a) (b)

Figure 3.13: Parallel multi-instance send tasks (a) and its macro expansion (b).

rules. The parallel case, instead, is more tricky. It is commonly considered as a macro as well: the parallel multi-instance task is thought of as a set of tasks between AND split and join gateways [25, 80], 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, thus requiring a direct definition of its formal semantics. In fact, consider for example the collaboration fragment in Figure 3.13 (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 obtain the collaboration fragment in Figure 3.13 (b), which however is not semantically equivalent. Indeed, each instance of *ORG B* in Figure 3.13 (a) has a *Task B* that sends only one message, while in Figure 3.13 (b) each instance has a *Task B* sending three times the same message, one for each copy of *Task A* in *ORG A*. This suggests that parallel multi-instance tasks are not simple macros, but they require their own direct formalization, as done in this thesis.

3.5 Comparing with other Approaches

This section discusses the most relevant attempts at formalizing the semantics of BPMN collaborations in the presence of multiple instances and data. The rest of the section focuses on the formalization of the more intricate aspects of the semantics.

3.5.1 On Formalizing Multiple Instances and Data

Many works in the literature attempted to formalize the core features of BPMN. However, most of them (see, e.g., [21, 18, 16, 25, 23, 81, 8, 73]) do not consider multiple instances and data, which are the focus of this work. Considering these features in BPMN collaborations, relevant works are [47, 49, 41, 27]. Meyer et al. in [47] focus on process models where data objects are shared entities and the correlation mechanism is used to distinguish and refer to data object instances. The use of data objects local to (multiple) instances, exchange of messages between participants, and correlation of messages are instead the focus of this thesis. In [49], the authors describe a model-driven approach for BPMN to include the data perspective. Differently from us, they do not provide a formal semantics for BPMN multiple instances. Moreover, they do not use data in decision gateways. Moreover, Kheldoun et al. propose in [41] a formal semantics of BPMN covering features such as message-exchange, cancellation, multiple instantiation of sub-processes and exception handling, while taking into account data flow aspects. However, they do not consider multi-instance pools and do not address the correlation issue. Semantics of data objects and their use in decision gateways is instead proposed by El-Saber and Boronat in [27]. Differently from the proposal, this formal treatment does not include collaborations and, hence, exchange of messages and multiple instances. Considering other modeling languages, YAWL [80] and high-level Petri nets [69] provide direct support for the multiple instance patterns. However, they lack support for handling data. In both cases, process instances are characterized by their identities, rather than by the values of their data, which are however necessary to correlate messages to running instances.

Regarding choreographies, relevant works are [28, 42, 32]. López et al. [28] study the choreography problem derived from the synchronization of multiple instances necessary for the management of data dependencies. Knuplesch et al. [42] introduce a data-aware collaboration approach including formal correctness criteria. However, they define the data perspective using data-aware interaction nets, a proprietary notation, instead of the wider accepted BPMN. Improving data-awareness and data-related capabilities for choreographies is the goal of Hahn et al. [32]. They propose a way to unify the data flow across participants with the data flow inside a participant. The scope of data objects is global to the overall choreography, while the proposed work considers data objects with scope local to participant instances, as prescribed by the BPMN standard. Apart from the specific differences mentioned above, our work differs from the others for the focus on collaboration diagrams, rather than on choreographies. This allows to specifically deal with multiple process instantiation and messages correlation.

Concerning the correlation mechanism, the BPMN standard and, hence, this thesis have been mainly inspired by works in the area of service-oriented computing (see the relationship between BPMN and WS-BPEL [52] in [53, 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 others, the COWS [57] formalism captures the basic aspects of service-oriented systems, and in particular service instantiation and message correlation à la WS-BPEL. From the formal point of view, correlation is realized by means of a pattern-matching function similar to that used in the provided formal semantics. Let us focus more on how correlation is dealt with in BPMN [53, Sec. 8.3.2]. The standard identifies two mechanisms to manage the correlation of messages with process instances. The first is a *key-based* mechanism that couples sender and receiver by means of the concept of correlation key. Any message, to be properly correlated, needs to carry values of a correlation key within its payload. Those values are initialized during the first interaction and then extracted, even partially, to correctly match the follow-up messages. The second is instead a *context-based* mechanism, as it depends on the process data (i.e., the content of data elements) associated to the process instances. This is a more expressive form of correlation with respect to key-based correlation, since this latter can only populate a correlation key implicitly from the values of the first message. Instead, in this case a correlation key can contain formal expressions dynamically evaluated at run-time using the process context, hence the correlation key can be automatically updated whenever the underlying data elements change.

"In that sense, changes in the Process context can alter the correlation condition" [53, p. 75].

Similarly, the presented formalization defines correlation keys in the receiving elements to match the correct messages and to store their payload. In particular, in this case correlation keys are identified by the template fields that are not formal (i.e., without the ?-tag). Since non formal fields are either values or expressions, the proposed approach based on pattern-matching is able to mimic both the key-based and the context-based mechanism. However, BPMN considers a correlation key as

"a composite key out of one or many CorrelationProperties that essentially specify extraction Expressions atop Messages" and, in the context-base case, *"atop the Process context"* [53, pp. 75-76].

Thus, correlation properties can be quite articulate expressions, especially when they have to be used to match messages with complex structures. Instead, in this case, messages have a simple tuple structure, i.e., they are ordered lists of values. As a consequence, templates are ordered lists as well, and the correlation mechanism, once the template expressions have been evaluated, simply performs a (field-by-field) pattern-matching check.

Concerning data-awareness in process modeling, several works refer to the data-centric approach (see, e.g., the surveys [59, 11, 34]). This approach uses data elements as first class citizens and focuses on their life cycles (i.e., on the data flow) [48]. Differently, the proposed approach focuses on BPMN as reference language, thus concentrating on activities and, more generally, on control flow. Data elements act as pre- and post-conditions for activity execution, and as main decision indicator at exclusive gateways. Moreover, the main interest of this thesis is the study of the BPMN management of multiple instances that, even if it is affected by data, keeps the focus on the control flow perspective.

Finally, to the best of our knowledge, no work in the literature permits to specify different execution modalities (i.e., atomic, non-atomic concurrent, non-atomic non-concurrent) for tasks of BPMN models. This feature allows to study, e.g., the impact of different settings of such modalities in BPMN models involving multi-instance tasks that access data.

3.5.2 On Formalizing the OR-Join Gateway

Most of the previous attempts to formalize the semantics of the OR Join [76, 26, 65, 12] are based on earlier versions of the BPMN standard, which provides different semantics for the OR Join. Moreover, also when the same version of the standard is considered, different interpretations of the OR Join behavior, not always faithful to the specification, have been given. In particular, these differences regard the treatment of mutually dependent OR Joins (the so-called 'vicious circles') and of deadlock upstream an OR Join. In fact, from a faithful translation of the standard, it results that mutually dependent OR Joins are blocked, and that an OR Join is not able to recognize that there is a deadlock on a path leading to it, thus it will wait forever. Below, we discuss the most significant related works.

Volzer [76] proposes a non-local semantics for the OR Join in the BPMN 1.0 specification (2006) using workflow graphs. In case of vicious circles he argues that the intended meaning is not clear and hence they should be sorted out by static analysis. This approach is then improved in [77], which quotes the 2010 version of the specification and gives an informal description of this one by means of inhibiting and anti-inhibiting paths. Dumas et al. [26] base their work on BPMN 1.0 and on the definition of the Synchronisation Merge pattern to which the specification refers to. They provide a local semantics, without imposing restrictions on the language, able to detect deadlocks upstream and to unlock mutually dependent OR Joins. Thalheim et al. [65] make use of ASMs to introduce the OR Join, by referring to the the specification of 2006, and make a comparison between the definitions given by other authors. Adopting a token-based view of workflow semantics, they start to analyze acyclic models. In this case, to treat the

OR Join, they introduce a special type of synchronization tokens that fire flow objects in their downstream. They then consider cycles and, to deal with synchronization in their presence, they introduce sets of tokens, which are viewed as a coherent group when a join fires. Carbone et al. [12] refer to BPMN 2.0 - Beta 1, providing a global semantics directly in terms of a subset of BPMN. As for the vicious circles, they argue that, since informally the BPMN specification does not include the resolution strategy and their work is a faithful translation, they do not consider it.

Differently from this thesis, the above approaches rely on past versions of the BPMN standard, which provide different semantics for the OR Join with respect to the current 2.0 version. Thus, they cannot be applied as they are to the standard BPMN 2.0. Moreover, concerning the issues about vicious circles and deadlocks upstream considered by some of those works, we have checked how they are dealt with by the current specification and, to be completely faithful with it, we have simply applied the same solution. Indeed, in the current description of the OR Join semantics (Figure 3.7), there does not seem to be any ambiguity about these two issues. The OR Join is able to detect neither a vicious circle nor a deadlock upstream, thus in both cases its execution is blocked forever.

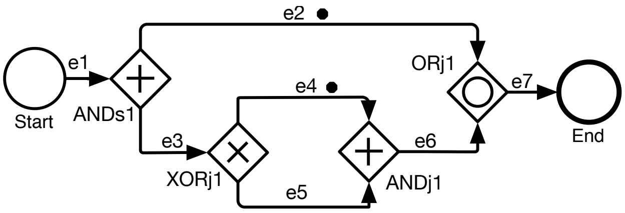


Figure 3.14: OR Join with deadlock upstream.

As an example, consider the situation depicted in Figure 3.14, where ANDj1 is deadlocked because no token will ever arrive in e_5 to synchronize with the one in e_4. The OR Join activation condition “At least one incoming Sequence Flow has at least one token” is satisfied, as there is a token in e_2. Regarding the other condition, the sequence edge f can only be e_4; now, for the path formed by e_4 and e_6 there is no other path starting with e_4 and ending with a marked edge incoming to the OR Join (like e_2). Thus, the OR Join is not activated and will never be, as the waiting token in e_4 will never move.

Recently, Prinz et al. [56] propose a formalization of the OR Join semantics referred to the current version of the standard. However, they limit the work to *sound* workflow graphs, which identify a quite restricted class of BPMN processes [83]. In fact *soundness* is defined as the combination of properties concerning the dynamic behavior of a process: option-to-complete, proper-completion, and no-dead-activities. Moreover, the proposed semantics do not fit with the standard as, for instance, it avoids vicious circles by determining which OR Join in a circle has to wait and which one must proceed.

PART III

FORMALIZATION AT WORK

CHAPTER 4

MIDA: MULTIPLE INSTANCES AND DATA ANIMATOR

The formal semantics introduced so far shows how complex the modeling of a collaboration diagram can be, and how to properly figure out the interplay between multiple-instances, messages and data is in general an error-prone and time-consuming task. Business process animation plays an important role in facing the understanding of a multi-instance collaboration. Indeed, the dynamic visualization of business processes through animation allows to control the execution of multiple process instances at the same time thanks to graphical effects. Moreover, the animation can facilitate the modeling and comprehension of other intricate situations, such as the complex management of the correlation mechanism or the handling of data and messages.

This chapter proposes a tool for animating BPMN diagrams by means of 2D token flow. The rest of the chapter presents the tool and shows how it can effectively support designers in debugging their models. The description of the tool's functionalities resorts to the running example in Figure 2.13.

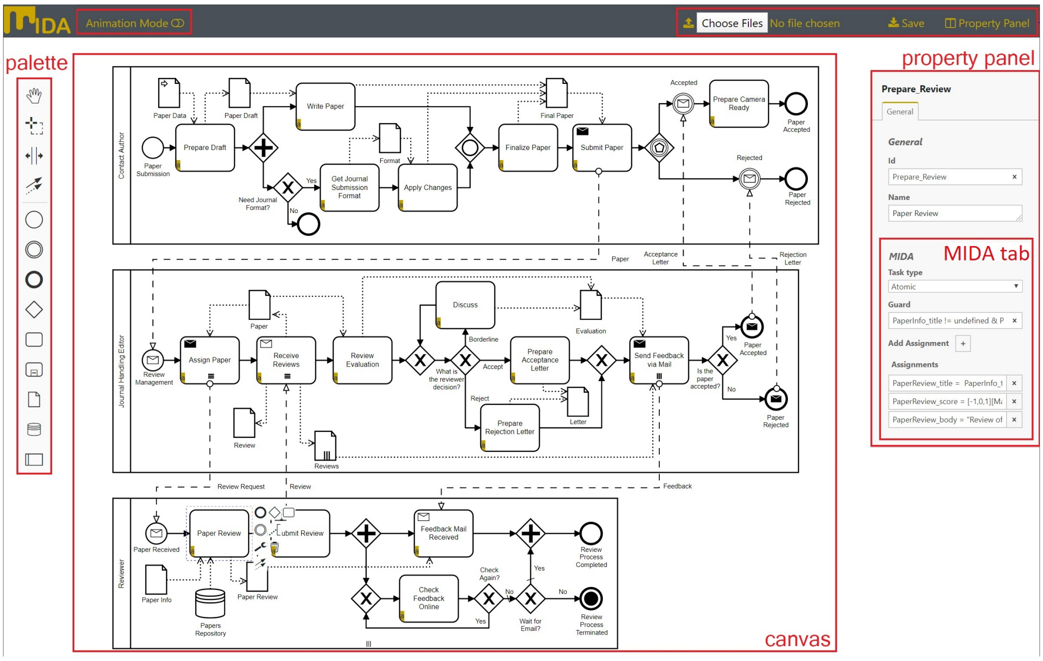
4.1 MIDA Overview

MIDA (*Multiple Instances and Data Animator*), is an animator tool that supports designers in achieving a more precise understanding of the behavior of a collaboration diagram by means of the visualization of the model's execution, also in terms of the evolution of the values of data objects and messages. MIDA animation features result helpful both in *educational contexts*, for explaining the behavior of BPMN elements, and in practical modeling activities, for *debugging* errors that can easily arise in multi-instance collaborations.

Technically speaking, MIDA is a web application written in JavaScript, it is accessible online by users via a web browser without installing any software, or it can be ran locally using the executable version available for Windows, OSX, and Linux systems. MIDA has been realized by extending the *“bpmn.io Token Simulation”* plugin by Camunda [55]. The tool as well as its source code, binaries, a tutorial and example models are freely available at `http://pros.unicam.it/mida`. MIDA can be redistributed and/or modified under the terms of the MIT License.

Concerning the user interface, MIDA provides two main views, one for the modeling and one for the animation. Both of them are composed by a top navigation menu and a *visualization* area.

Launching the application, the *modeling interface* is shown. It embeds a user-friendly BPMN modeling environment, by which to design the process diagrams, and the menu where to find the main functionalities, see Figure 4.1. More in detail, the elements composing the modeling interface are:



Prepare_Review

General

General

Id: Prepare_Review

Name: Paper Review

MIDA

Task type: Atomic

Guard: PaperInfo_title != undefined & P

Add Assignment +

Assignments

PaperReview_title = PaperInfo_t

PaperReview_score = [-1,0,1][Ma

PaperReview_body = "Review of

Figure 4.1: MIDA modeling interface.

- the **animation mode** toggle, that permits to switch from the modeling interface to the animation one, and vice-versa;
- the **upload** button to load models previously designed;
- the **save** button, that permits to locally save models in the standard format *.bpmn*;

- the **property panel** including the MIDA *tab*, to specify attributes of the BPMN elements contained in the diagram;
- the **canvas**, where BPMN elements are placed to form a collaboration diagram; and
- the **palette**, where to drag and drop elements in the diagram.

Differently, by switching to the animation mode, the interface changes a bit by disabling the modeling functionalities, and enabling the ones for animation. More specifically, the animation interface, Figure 4.2, is composed of:

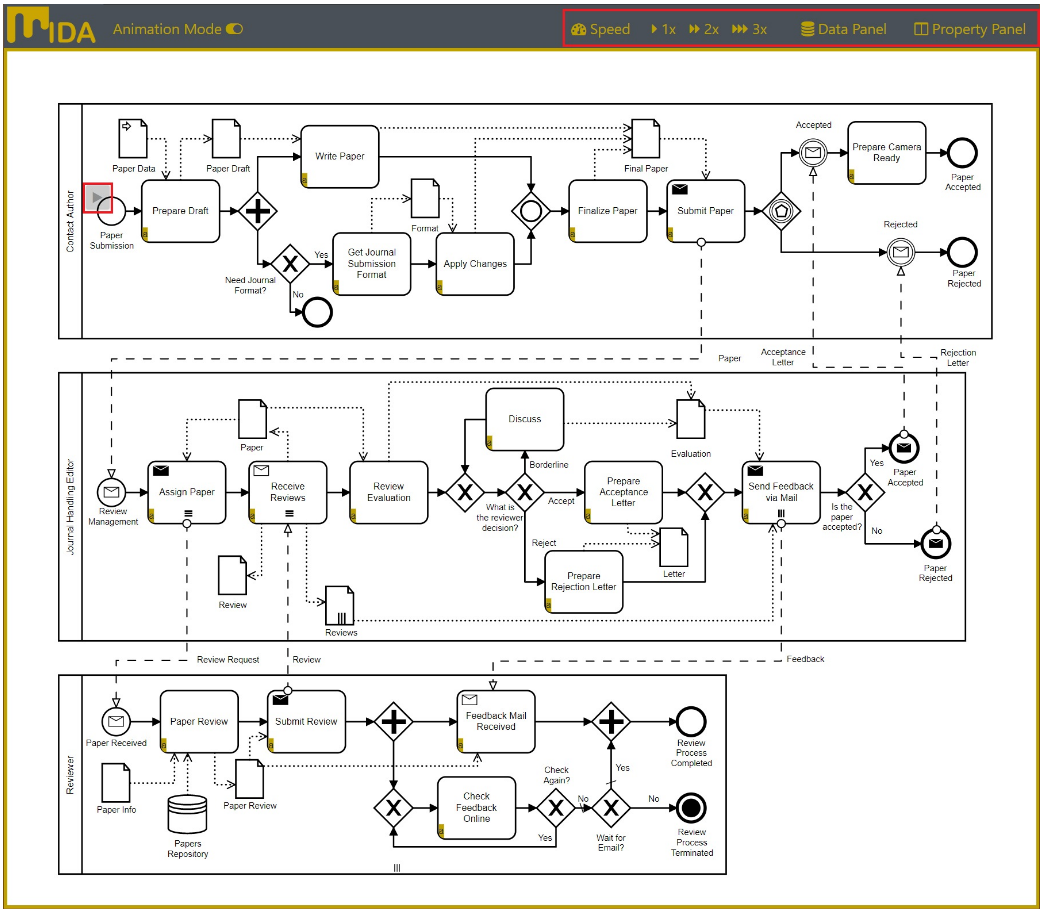


Figure 4.2: MIDA animation interface.

- the **animation mode** toggle, that allows to switch back to the modeling interface;
- the **speed** choicer for increasing or decreasing the speed of the animation;

- the animation **controls** that permit to pause or reset the animation, and open a pop-up panel showing all the events that happened during the animation; and
- the **data panel** for monitoring the evolution of values associated to data items.

4.2 MIDA **in Action: Modeling**

The starting point to exploit the MIDA functionalities is the modeling of a BPMN diagram by means of the MIDA modeling environment. This can be done starting from an empty diagram or loading a *.bpmn* file. The palette on the left side of the interface permits to insert new elements like tasks or gateways and choose their specialization such as a send task or a xor gateway. A complete reference on the usage of the modeler can be found here at the *bpmn.io* website [1].

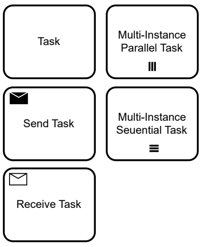
Notably, the design goes beyond the graphical representation of the collaboration diagram. The standard *.bpmn* file format stores various information of model elements. Those are essential for modeling BPMN diagrams. However, capturing all the aspects described by our semantics requires to extend the XML.

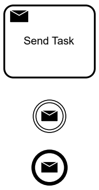
The MIDA modeler uses the *extension element* tag to include additional information, regarding execution modality, data, and message contents, in the *.bpmn* file, keeping intact the compatibility with the other modeling tools. Table 4.1 shows the elements which XML representation has been extended and the corresponding fragment. Activities embed into their extension elements *attributes* specifying the execution modality, the guard, and the assignments (i.e., respectively the <mida:type>, <mida:guard>, and <mida:assignments> tags). Sending and receiving elements uses the <mida:message> tag to provide message related information by means of fields. Finally, data elements specifies variables through the <mida:dataObjFields> tag.

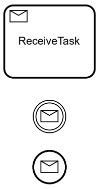
In this regard, the property panel plays a key role when modeling collaborations with MIDA as it permits exploiting XML attributes of the *.bpmn* format to specify and save information about the BPMN elements. It provides several fields where to insert information that regards the selected element. Apart from a subset of fields that is in common with all the elements (i.e., *id* and *name*), different information needs to be provided depending on the considered BPMN element. In particular, information about multi-instance characteristics, data elements, and messages. Such information, which represents the peculiarities of the proposed formal semantics, are introduced

[1] `https://bpmn.io`

Element	Extension Element

 Task
Multi-Instance Parallel Task
Send Task
Multi-Instance Seuential Task
Receive Task
	<bpmn:extensionElements>
	  <mida:taskType>
	    <mida:type>a</mida:type>
	  </mida:taskType>
	  <mida:guard>
	    <mida:exp>x == 0</mida:exp>
	  </mida:guard>
	  <mida:assignments>
	    <mida:assignment assignment="y = 2" />
	    <mida:assignment assignment="y = x" />
	  </mida:assignments>
	</bpmn:extensionElements>

	<bpmn:extensionElements>
	  <mida:message>
	    <mida:field field="10" />
	    <mida:field field="a" />
	  </mida:message>
	</bpmn:extensionElements>

	<bpmn:extensionElements>
	  <mida:message>
	    <mida:field field="x" />
	    <mida:field field="y"
	      isCorrelation="true" />
	    <mida:field field="z" />
	  </mida:message>
	</bpmn:extensionElements>

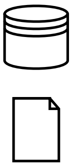
	<bpmn:extensionElements>
	  <mida:dataObjFields>
	    <mida:dataField dataField="x = 1" />
	    <mida:dataField dataField="y = 'Text'" />
	    <mida:dataField dataField="z" />
	  </mida:dataObjFields>
	</bpmn:extensionElements>

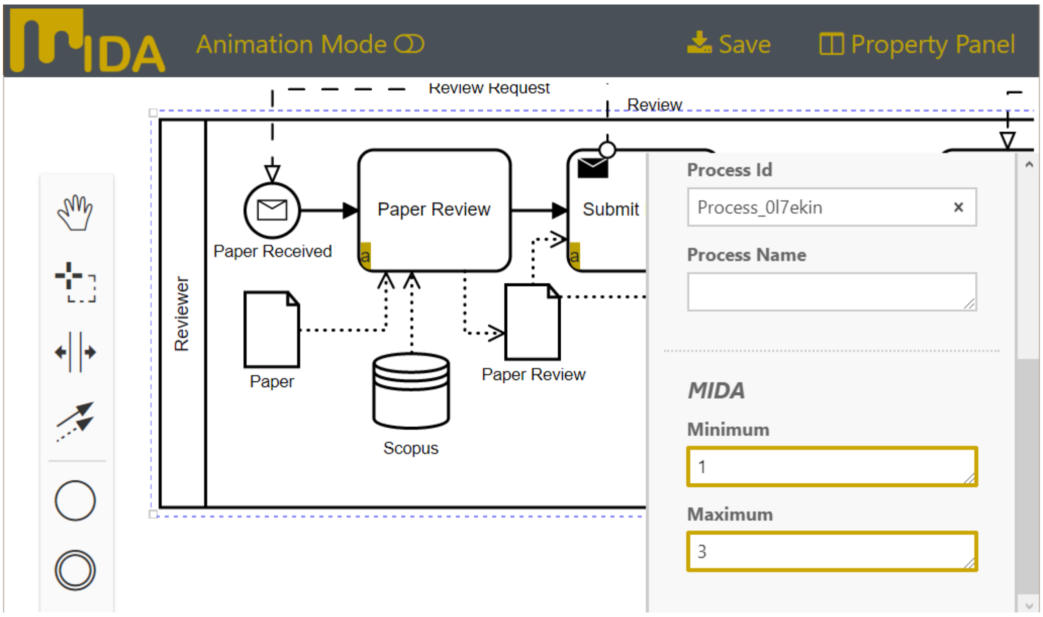
Table 4.1: MIDA extension elements correspondence.

during the design of the model by means of the MIDA tab contained in the *Property Panel*. Following, it is shown how to specify this information in MIDA using the paper reviewing collaboration of Figure 2.13 as example.

Multi-instance characteristics. This information affects both pools and tasks, identifying how many repetitions of the inner process or the activity have to be done. By selecting a pool element in the collaboration diagram, Figure 4.3, two input fields named *Minimum* and *Maximum* become available. They allow to constrain the number of instances that will be executed for that pool. In the running example, the *Reviewer* pool has *Minimum* =

1 and $Maximum = 3$, hence every time a *ReviewRequest* message is received by the pool, it triggers the activation of one instance. Three instances at most will be activated for the *Reviewer* pool.

Differently, to model a multi-instance task one has to firstly insert a new task in the diagram dragging it from the palette. Then, the needed



Process Id
Process_0l7ekin ×

Process Name

MIDA

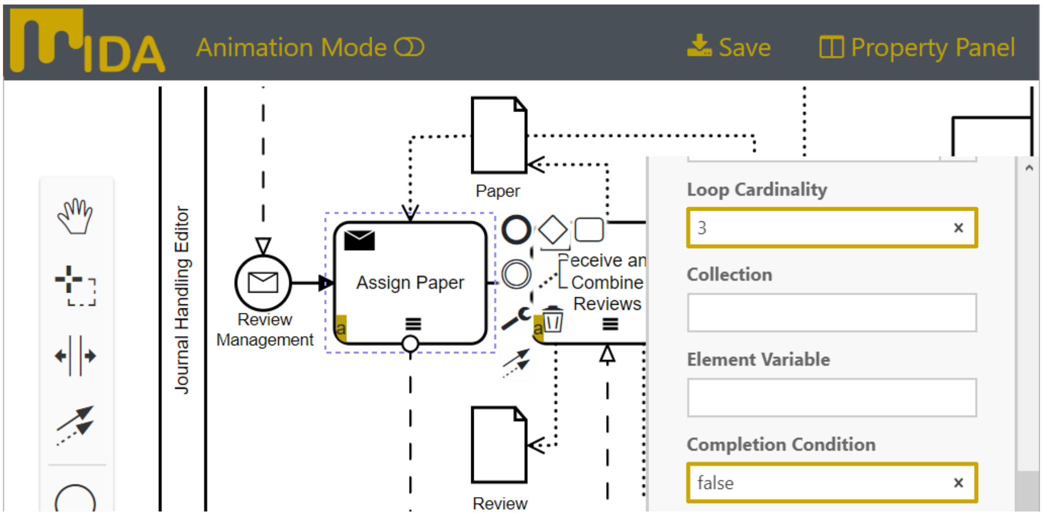
Minimum
1

Maximum
3

Figure 4.3: Multi-instance pool modeling.

multi-instance marker (||| or ≡) has to be chosen from the element context pad. At this point, the MIDA *tab* provides two input fields where to express the *loopCardinality* and the *completionCondition* of the task, see Figure 4.4. These two parameters can be either a concrete value, for instance $loopCardinality = 5$ and $completionCondition = \mathsf{true}$, or expressions such as $\mathsf{x} + \mathsf{y}$ and $\mathsf{x} > 0$. However, the evaluation of these input fields indicates, respectively, the number of task instances to be executed and the condition for an early termination of the multi-instance task. For example, the parallel multi-instance *AssignPaper* send task in the running example has $loopCardinality = 3$ and $completionCondition = \mathsf{false}$, meaning that there will be exactly three parallel executions of the task.

Data element variables. Data elements specify which are the variables that have a role in the collaboration, formally they are structured in fields, so that to mention a variable one needs to use the syntax do.field, where the dot is a fields operator. Since MIDA is developed in *Javascript*, it exploits the same syntax in order to specify data element fields, as well as conditions, expressions, and so on. Moreover, MIDA exploits the *Javascript* capability of evaluating at run time scripts written in this scripting language. Thus,



Loop Cardinality: 3

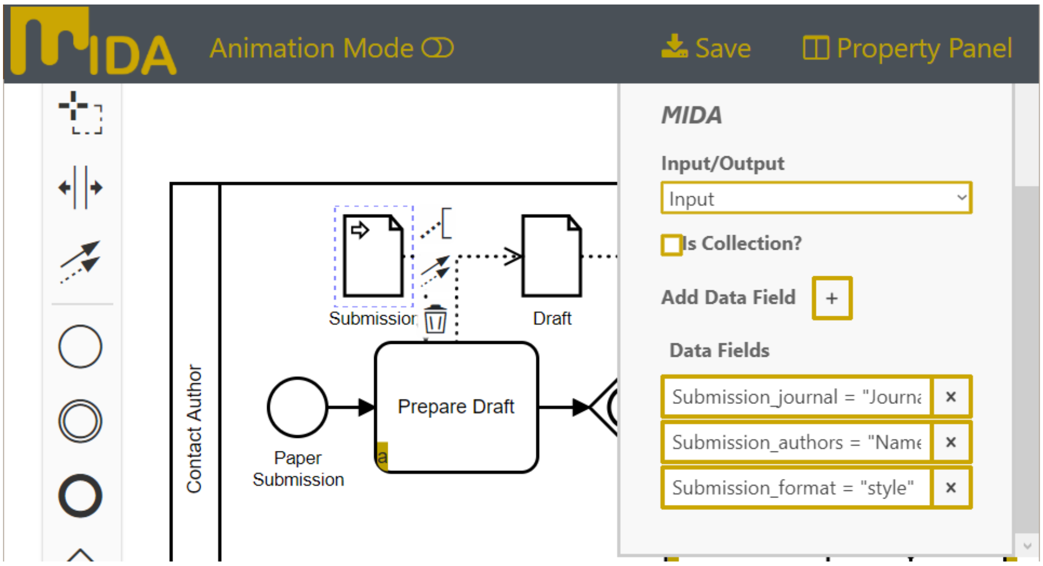
Collection:

Element Variable:

Completion Condition: false

Figure 4.4: Multi-instance task modeling.

one cannot use the dot symbol to mean the access to the fields of a data element, because it serves as member operator. To overcome this problem, one can alternatively use the _ symbol (e.g., do_field). Anyway, fields in data



MIDA

Input/Output: Input

□ Is Collection?

Add Data Field +

Data Fields

Submission_journal = "Journa

Submission_authors = "Name

Submission_format = "style"

Figure 4.5: Data elements fields modeling.

elements (i.e., data objects, data stores, data collections, data input/output) are specified in the same way. They can be added one by one from the MIDA *tab* pressing the *Add data field* button which lets a new input text appear, see Figure 4.5. The designer can write the name of the variable he/she needs. Due to the use of *Javascript*, that is an untyped language, data element fields can hold a value of any data type. They can be initialized (e.g., *PaperData_journal* = 'Journal Name'), in order to specify data inputs, or be undefined (e.g., *PaperDraft_title*).

Using the MIDA *tab*, users can also select among a simple data object, input/output data object or data object collection. Notably, a data collection represents a stack of data items that can be created, retrieved, and reinserted. As prescribed by the formal semantics, MIDA provides dedicated functions to support such features. For instance, given a data collection named *Collection* with two fields *first* and *second*, the designer can:

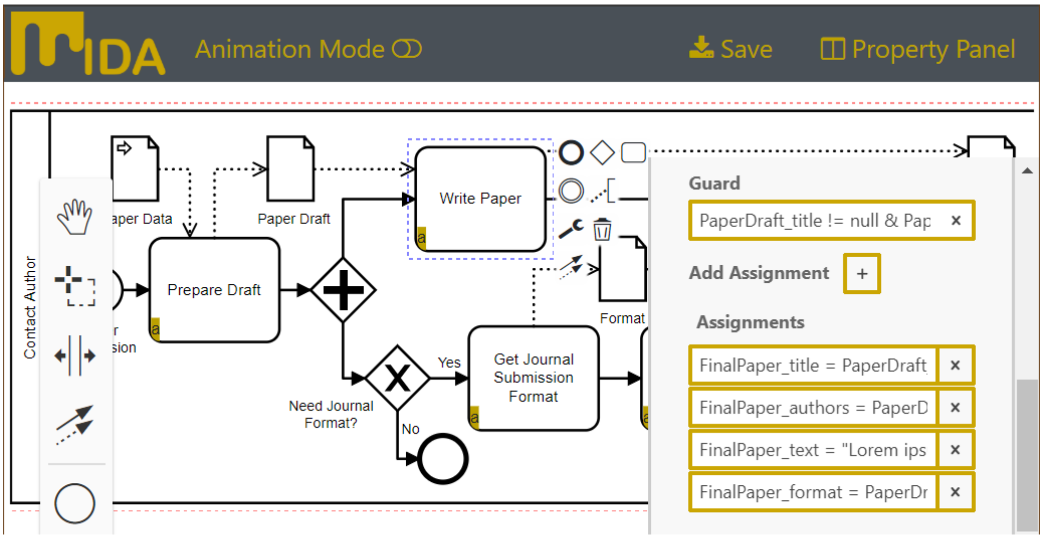
- **create** a new item for the collection by means of function createItem(Collection). Once the new item is created, the designer can assign values or expressions to their fields (e.g., Collection_first = 9);

- **retrieve** the item on top of the collection via function getItem(Collection). This function makes available the two fields on top of the collection in order to change their values or use them in the right term of an assignment (e.g., DataObject_field = Collection_second); or

- **insert** an item on top of the collection using function putItem(Collection). This function requires the creation or the retrieving of an item to be subsequently pushed in the collection.

Variables manipulation. According to the BPMN standard, the access to data is represented by associations between data elements and tasks that can predicate over field names. These associations can define preconditions for the execution of a task, expressed in MIDA as a task *guard*. On the other hand, the effects of a task execution on a data element is instead specified by means of a list of *assignments*. Figure 4.6 shows the guard and the assignments related to the *WritePaper* task introduced in the *Contact Author* pool. Guards can be any kind of Boolean expression over data elements fields or concrete values using the standard operators (e.g., equal to ===, not equal to !=, lower than <, greater than >, lower than or equal to <=, greater than or equal to >=, etc.). Likewise for the assignments that use the standard operators (e.g., basic assignment =, addition assignment +=, subtraction assignment -=, multiplication assignment *=, division assignment /=, etc.). It is worth noting that the MIDA *tab* allows to change the execution modalities of a task. Task behavior can be chosen from *atomic*, *non-atomic concurrent*, or *non-atomic non-concurrent*. A golden label at the right-bottom corner of the task shows the acronym of the execution modality (i.e., a, na_c, na_nc). In the running example all tasks are atomic.

Message exchanges. Sharing information between participants is done by means of message exchange. Indeed, the presented semantics allows one to



Guard

PaperDraft_title != null & Pap ×

Add Assignment +

Assignments

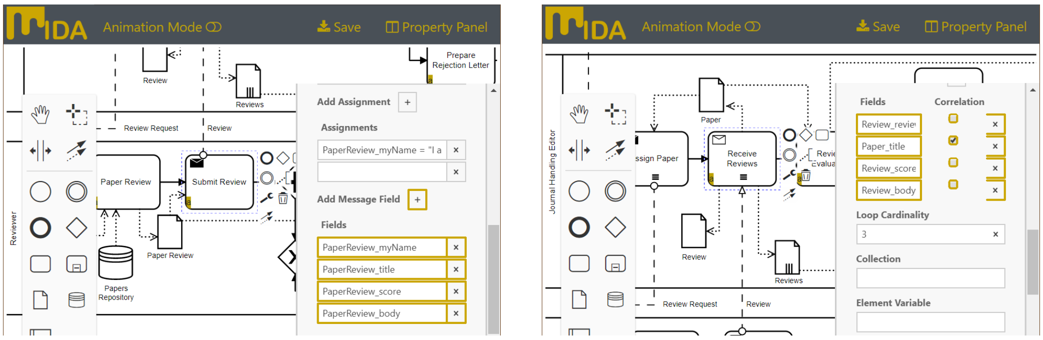
FinalPaper_title = PaperDraft ×

FinalPaper_authors = PaperD ×

FinalPaper_text = "Lorem ips ×

FinalPaper_format = PaperDr ×

Figure 4.6: Task's guard and assignments.



(a)

Add Assignment +

Assignments

PaperReview_myName = "I a ×

×

Add Message Field +

Fields

PaperReview_myName ×

PaperReview_title ×

PaperReview_score ×

PaperReview_body ×

(b)

Fields	Correlation	
Review_reviev	□	×
Paper_title	☑	×
Review_score	□	×
Review_body	□	×

Loop Cardinality

3 ×

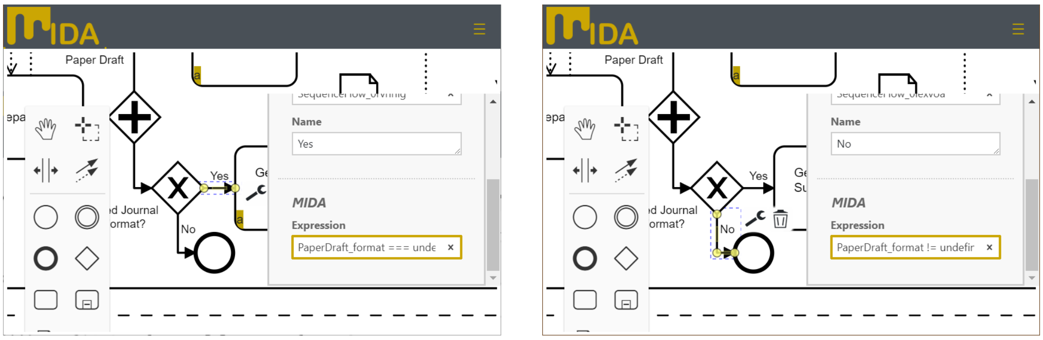
Collection

Element Variable

Figure 4.7: Message exchanges.

express both the content of the message to be delivered from the sender, and also which kind of messages the receiver wants to get. From the sender perspective, the designer specifies a tuple of concrete values and/or data element fields. This tuple is the payload of the message, and it is specified in the MIDA *tab* by a list of fields as shown in Figure 4.7 (a). On the other hand, the receiver can retrieve only the messages that matches particular requirements. This is expressed by a template, that is a tuple of data element fields. Also in this case this tuple is provided for all the receiving elements as separate input fields, see Figure 4.7 (b). In addition, for each message field there is also the possibility to put a tick on a correlation box in order to set the functioning of pattern matching. Fields with a tick are used to match the correct message, while those without a tick represent the fields that will take values from the received message.

Data driven gateways. Values stored in data elements can also be used by conditions associated to the outgoing sequence edges of XOR and OR split gateways, in order to support decisions. Conditions have to be specified in the input field of the MIDA *tab* that appear selecting each outgoing sequence flow of the decision gateway. Figure 4.8 (a) shows the conditional expression contained into the *Yes* branch of the XOR gateway in the *Contact Author* process of the running example that is PaperDraft_format === undefined. It checks if the task *Prepare Draft* in the pool *Contact Author* has set any value in the field *format* of *Paper Draft*, and consequently drives the process execution to the following task that is *Get Journal Submission Format*. While Figure 4.8 (b) shows the conditional expression contained into the *No* branch that is PaperDraft_format != undefined.



Name
Yes

MIDA
Expression
PaperDraft_format === unde ×

(a)

Name
No

MIDA
Expression
PaperDraft_format != undefir ×

(b)

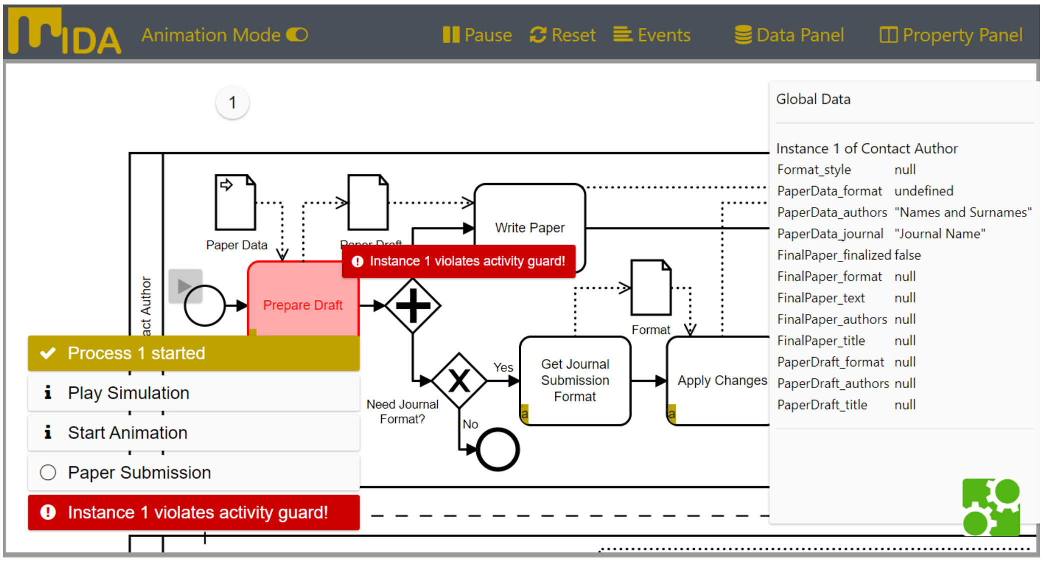
Figure 4.8: Decision gateways.

4.3 MIDA in Action: Animation and Debugging

The key characteristic of MIDA is the animation of collaboration models, which enables models debugging, supporting the visualization of data evolution and multiple instances execution. Anyway, like in software code debugging, the identification and fixing of bugs are still in charge of the human user.

By selecting the *Animation Mode* button in the modeling interface of MIDA, Figure 4.1, the element palette disappears, as well as the load and save buttons to leave space for the animation controls.

Once in animation mode, MIDA depicts a *play* button over each fireable start event, see Figure 4.2. Every time this button is clicked, a new instance of the corresponding process is activated, accordingly with the multi-instance constraints specified in the modeling phase. Graphically, this correspond to the creation of a new token labeled by a fresh instance identifier. Process tokens in MIDA are golden circles containing their instance identifiers, tokens with the same identifier corresponds to parallel executions of the same process instance. Notably, to show the delivery of a message, *message* tokens cross the message flows. Message tokens are green circles with an identifier equal to the one of the process instance throwing the message. Then, as shown in Figure 4.9, the token starts to cross the model according to the operational rules induced by the formal semantics. The animation terminates once no token can move forward, since no semantic rule can be applied.



Instance 1 violates activity guard!

✓ Process 1 started

ℹ Play Simulation

ℹ Start Animation

○ Paper Submission

Instance 1 violates activity guard!

Global Data

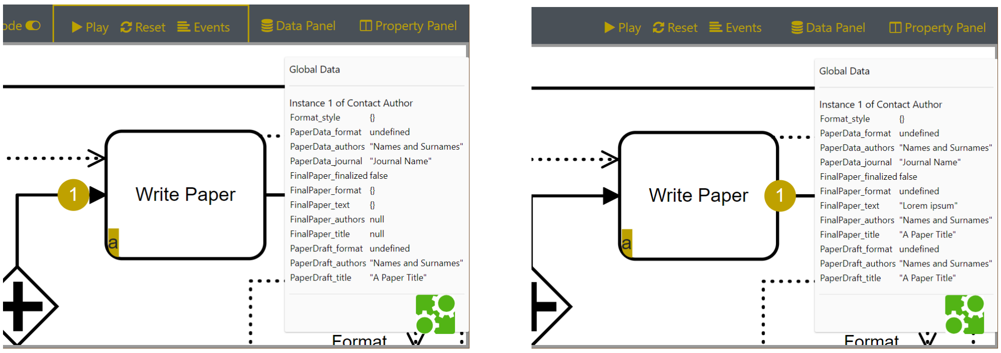
Instance 1 of Contact Author

Format_style	null
PaperData_format	undefined
PaperData_authors	"Names and Surnames"
PaperData_journal	"Journal Name"
FinalPaper_finalized	false
FinalPaper_format	null
FinalPaper_text	null
FinalPaper_authors	null
FinalPaper_title	null
PaperDraft_format	null
PaperDraft_authors	null
PaperDraft_title	null

Figure 4.9: MIDA in action.

From the Data Panel, users can monitor the evolution of the data state function σ_d of each process instance, by observing the values associated by the function to data element fields, which are organized according to the process instances they belong to. Figure 4.10 shows how data values change after the execution of the task *WritePaper* by means of the application of rules $P\text{-}Task_A$. This rule requires to satisfy the guard condition associated to the task and perform the assignments. The guard condition, PaperDraft_title != undefined & PaperDraft_authors != undefined, checks that the paper draft has been created, while the assignments populate the fields of *Final Paper* data object. Indeed, fields *title*, *authors*, and *format* of *Final Paper* are filled with the values of the homonym fields of data object *Paper Draft* (e.g., FinalPaper_title = PaperDraft_title), while field *text* is populated with a custom string.

Debugging. MIDA can effectively support designers in identifying issues in their business collaborations that can be related to bad modeling or a misunderstanding of the BPMN semantics. Indeed, the animation permits



(a)

Global Data

Instance 1 of Contact Author

Format_style	{}
PaperData_format	undefined
PaperData_authors	"Names and Surnames"
PaperData_journal	"Journal Name"
FinalPaper_finalized	false
FinalPaper_format	{}
FinalPaper_text	{}
FinalPaper_authors	null
FinalPaper_title	null
PaperDraft_format	undefined
PaperDraft_authors	"Names and Surnames"
PaperDraft_title	"A Paper Title"

(b)

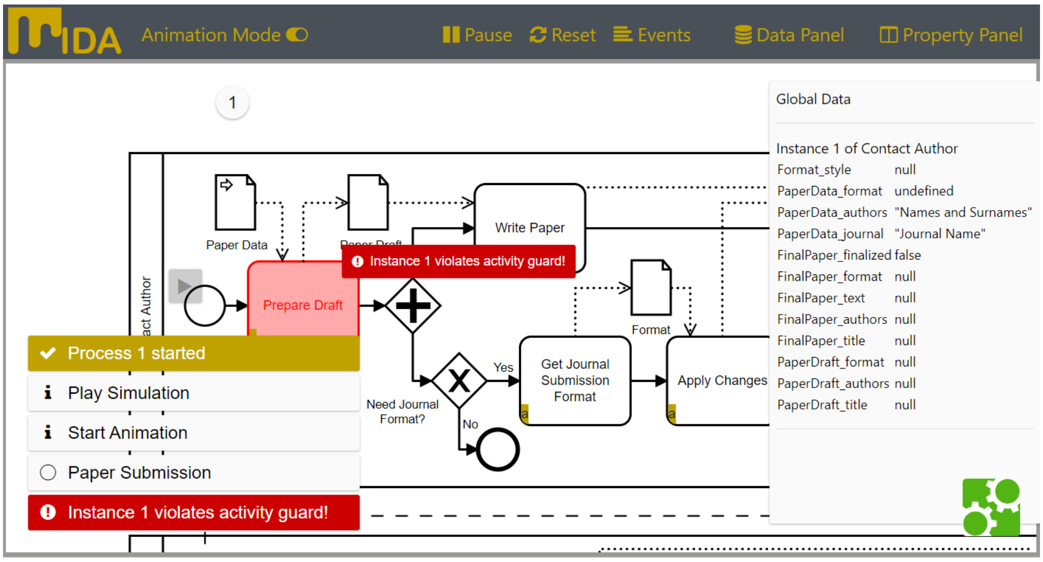
Global Data

Instance 1 of Contact Author

Format_style	{}
PaperData_format	undefined
PaperData_authors	"Names and Surnames"
PaperData_journal	"Journal Name"
FinalPaper_finalized	false
FinalPaper_format	undefined
FinalPaper_text	"Lorem ipsum"
FinalPaper_authors	"Names and Surnames"
FinalPaper_title	"A Paper Title"
PaperDraft_format	undefined
PaperDraft_authors	"Names and Surnames"
PaperDraft_title	"A Paper Title"

Figure 4.10: Data panel before (a) and after (b) the execution of task *Write Paper*.

to control the spread of tokens, hence the running activities, the spread of messages, and the evolution of the data elements fields. By pausing the animation, the designer can carefully observe the distribution of both process and message tokens, and at the values taken by data element fields in the data panel in order to immediately detect unwanted behavior. That information is strictly related to the provided semantics, representing respectively the state functions σ_e, σ_m, and σ_d. Moreover, if a token remains blocked or violates conditions (e.g., guard conditions, XOR/OR split conditions, constraints about maximum number of instances), MIDA highlights it in red, as shown in Figure 4.11 where the violation comes from the presence of an undefined field in the *Paper Data* data input. Considering the running example,



Instance 1 violates activity guard!

✓ Process 1 started

ℹ Play Simulation

ℹ Start Animation

○ Paper Submission

Instance 1 violates activity guard!

Global Data

Instance 1 of Contact Author

Format_style	null
PaperData_format	undefined
PaperData_authors	"Names and Surnames"
PaperData_journal	"Journal Name"
FinalPaper_finalized	false
FinalPaper_format	null
FinalPaper_text	null
FinalPaper_authors	null
FinalPaper_title	null
PaperDraft_format	null
PaperDraft_authors	null
PaperDraft_title	null

Figure 4.11: Guard violation.

receiving a *ReviewRequest* message triggers the activation of a new process instance of *Reviewer* rules *P-StartRcv* in combination with *C-CreateMi* , while a *Feedback* message has to be routed to an already existing *Reviewer* instance rules *P-TaskRcv*$_A$ in combination with *C-ReceiveMi* . Hence, in the latter case, the message needs to be properly correlated to the right instance. Otherwise, the *Journal Handling Editor* risks to send information to the wrong person. To ensure the correct correlation, the *Review* message, sent by each reviewer to the editor, contains a field representing the name of the reviewer that one can consider a unique identifier for the *Reviewer* instances. These names are then stored in the data collection *Reviews*, where each item of the collection contains a review for the same paper. Then, the task *Send Feedback via Mail* of the editor's pool retrieves the reviewer names and puts one of them in each of the payloads of the messages to send. Thus, thanks to the pattern-matching, each *Reviewer* instance will match the correct message comparing the field *name* of *Paper Review* with the one contained in the message payload. The same techniques can be applied in case of multiple submission of papers, so transforming the *Contact Author* pool to a multi-instance one. In order to do that, the editor has to associate to receiving a submission unique identifier that can be for instance the concatenation of the paper name and the authors. Then, once an acceptance or a rejection letter has to be delivered to a specific *Contact Author* instance, this identifier can be used to match the message. However, if the correlation check is not properly specified in the intermediate catch events *Accepted* and *Rejected* (e.g., no correlation data is provided), letters with the reply of the editor can be received by the wrong author. This possibly results with a review different from the correct one. However, MIDA allows to detect, and hence solve, this correlation issue.

Similarly, malformed or unexpected messages may introduce deadlocks in the execution flow, which can be easily identified by looking for blocked tokens in the animation. More in detail, MIDA depicts a red circle over a BPMN element in case a token remains blocked. This depends on the impossibility to apply any semantic rule to this particular token. For instance, in the running example a *Paper* message without the *title* field would never be consumed by the start message event *Review Management* of the *Journal Handling Editor* pool, because the premise of rule *C-ReceiveMi* performing the *match* function check is not satisfied, thus making the rule not applicable, see Figure 4.12.

Finally, since the proposed animation is based on data elements fields, also issues due to bad data handling can be detected using MIDA. Let us suppose that the sequential multi-instance receive task *Receive Reviews* in the pool *Journal Handling Editor* has a wrong loop cardinality set to two (instead of three). Rule *P-MisTask*$_1$ activates the multiple instance task.

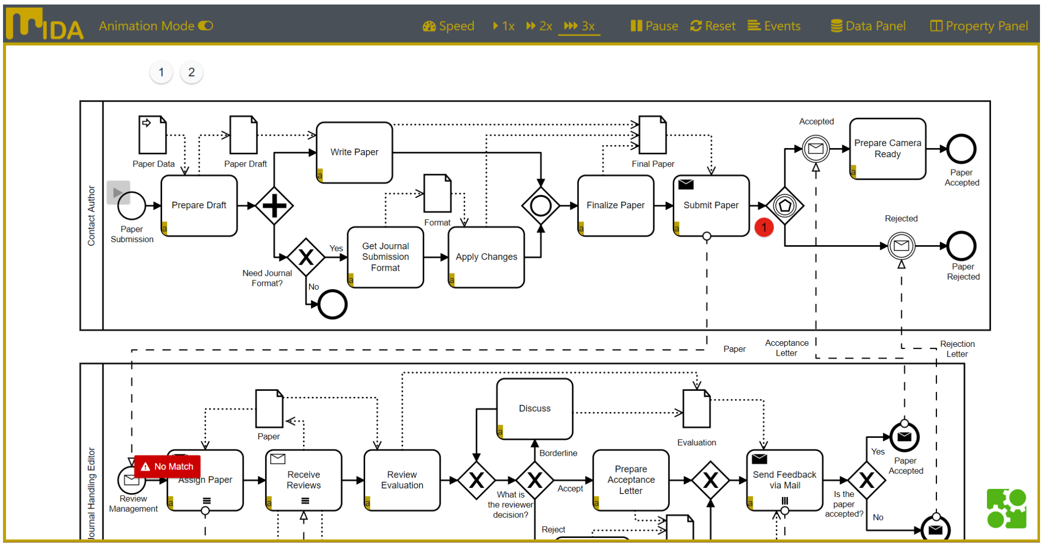


Figure 4.12: Wrong message arrival.

Then, after receiving two *Review* messages from the *Reviewers*, rules P-$TaskRcv_A$ and P-$MisTask_3$ can be applied, the two reviews are inserted as two items in the *Reviews* data collection, and the editor judges the paper with just two reviews. Finally, rules P-$MisTask_3$ completes the execution of multiple-instance tasks passing the token to the *Review Evaluation* task. However, once reached the task *Send Feedback via Mail*, the sending of three messages requires to retrieve three names from the data collection *Reviews*, but this cannot be applied, since the data collection contains just two items. This results in a deadlock easily detectable by means of MIDA.

To sum up, the MIDA tool can support designers in debugging their multi-instance collaboration models, as it permits to check the evolution of data, messages and processes marking while executing the models step-by-step.

4.4 Comparing with other Approaches

Scientific contributions concerning this topic are few, even if animation is recognized as a useful approach to enact the comprehension of business models. Some relevant contributions can be found in the literature, but also companies and modeling tool vendors provide some proposals. Differently from this work, in such implementations the interplay between multiple instances, messages and data is not fully supported.

For what concerns research papers focusing on the animation of business processes, Allweyer and Schweitzer [1] propose a tool for animating BPMN models that considers only processes, as it discards to animate message exchanges, both semantically and graphically. In addition, decision gateways do not depend on data conditions that the designer can specify a priori, but instead the tool requires that the user statically marks the path to choose. The authors permit to execute multiple instances of the same process, distinguishing them by the color of the tokens. However, the lack of data management does not allow to influence the behavior of the single process instances.

Aysolmaz [2] proposes an animator, PRIME, for BPMN process models decoupling the animation from the modeling that is not directly supported by the tool. PRIME animates process models using token flows upon the process model as in MIDA. It permits to watch continuous animations or step-wise ones of process models without handling message and data flow. Even though the graphical approach is notable, several BPMN concepts and modeling elements are not supported by the tool, and some of them have a wrong implementation. For instance, OR and Event-Based gateways behave in completely unpredictable ways, and End events act as Terminate End events killing any parallelization of the control flow.

About solutions for animation of BPMN models, some contributions come from the vendors of BPM tools. The animator of the Signavio [61] modeler allows users to step through the process element-by-element and to focus completely on the process flow. However, it discards important elements, such as message flows and data objects. Hence, Signavio animates only non-collaborative processes, without data-driven decisions, which instead are key features of this approach. Differently, Visual Paradigm [75] provides an animator that supports also collaboration diagrams. This tool allows users to visualize the flow of messages and implements the semantics of receiving tasks and events, but it does not animate data evolution and multiple instances. It is worth noting that animation via token flow affects also other fields of BPM since its capability simplifies the visualization of complex concepts. Indeed, the BPM tool Apromore [43] introduces the possibility to animate business processes via flows of tokens guided by event logs.

	[1]	[2]	[61]	[75]	[43]	MIDA
Multiple instances	✓	✗	✗	✗	✗	✓
Message exchange	✗	✗	✗	✓	✗	✓
Data	✗	✗	✗	✗	✗	✓
Data-driven gateways	✗	✗	✗	✗	✗	✓
Formal semantics	✗	✗	✗	✗	✗	✓

✓: supported, ✗: not supported

Table 4.2: Literature comparison.

Table 4.2 summarizes the features provided by the tools available in the literature and by MIDA. In detail, it compares the ability of: manage *multiple instantiating* of processes, animate *message exchange*, hence deal with BPMN collaborations, express *data* features, specify gateways *conditions* with data, and behave according to a formal semantics.

CHAPTER 5

UBBA: UNITY BASED BPMN ANIMATOR

The 2D animation approach proposed by MIDA can improve the understanding of the BPMN semantics and the quality of the produced BPMN diagrams. In the same fashion, this chapter investigates the possibility to exploit 3D graphical approaches for improving the stakeholders' experience when facing a BPMN model.

This chapter presents a tool that permits to create custom 3D virtual worlds from an input process model. Besides this 3D view of the diagram, the tool permits to see the model execution by means of an animation that faithfully follows the BPMN semantics. The chapter firstly provides an overview on the tool development and functioning. Finally, it concludes showing the tool in action using a collaboration diagram taken from the running example.

5.1 UBBA Overview

Business processes in the BPMN graphical notation are represented by means of static 2D flow charts, where the graphics of the elements embed their semantic categories (e.g., rectangles for activities, diamonds for choices, circles for events, etc.). This graphical representation is very straightforward for experts, but it may results difficult to understand by people who do not know the syntax and semantics of BPMN [9] (e.g., stakeholders). To this aim, BPMN animation tools can help model comprehension. However, animation approaches are mainly based on 2D diagrams, just few works investigate the use of a 3D world as an environment to closely portray the reality of the business process. Furthermore, when models become very large it is difficult to follow their execution semantics [46, 15] and the use of 2D instead of 3D representations limits the amount of information the user can perceive [78]. Indeed, to better explain the meaning of a flow element in a BPMN diagram, the designer can just make use of textual annotations. While turning a collaboration diagram into a virtual world, where 3D images replace textual information can be more effective.

Notably, the main objective of 2D animation is to facilitate the modeling and the debugging of BPMN collaborations. Differently, the 3D animation is intended to increase the comprehension of the business setting depicted in the diagram.

To overcome the above limitations, the prototype tool UBBA (Unity Based BPMN Animator) is proposed. Taken as input a BPMN collaboration diagram in standard XML format, UBBA recreates a custom virtual world where to visualize and to animate the input diagram decorated with 3D graphics chosen by the user. The user can follow the execution of its model looking at the animation on the newly created virtual world.

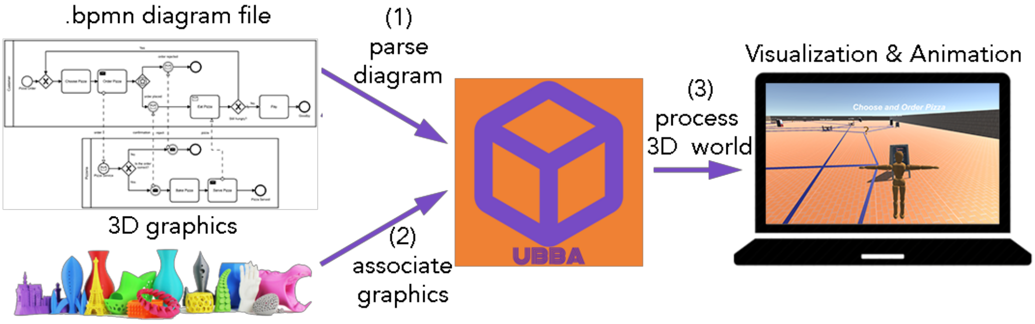


Figure 5.1: UBBA functioning.

As far as we know, UBBA is the first tool that can be exploited for representing and animating any kind of business scenario (e.g., order fulfillment, retrieval of healthcare data, bureaucratic procedures), thanks to its capability of loading standard BPMN XML files and custom 3D graphics. UBBA is conceived to support its users, i.e. business process designers, in:

- validating their BPMN diagram, possibly in collaboration with domain experts who help in choosing effective 3D graphics, and
- creating appealing 3D animations for stakeholders, who are the final audience for the products of the tool.

UBBA is a cross-platform and stand-alone tool that has been realized in Unity[1], a game engine for creating 2D/3D video-games or other interactive contents, such as architectural visualizations or animations in real time.

[1] www.unity3d.com

UBBA as well as its source code, binaries, guide, and a short demonstration video are available at `http://pros.unicam.it/ubba/`. UBBA can be redistributed and/or modified under the terms of the MIT License.

More in detail, UBBA aims at reproducing the setting described in a BPMN collaboration diagram and animating its execution, from the point of view of the resource, by means of token flow. Indeed, the BPMN elements are transformed into 3D graphics and visualized in a virtual space. Then, one or more tokens cross the diagram following the semantics of the BPMN elements they met. Figure 5.1 depicts the UBBA workflow functioning. In the following, the tool is introduced focusing on the visualization and then on the animation features.

The first characteristic of UBBA is the rendering of a BPMN model into a 3D space. To this aim, UBBA provides three main features:

- read a standard *.bpmn* file;
- associate a graphic to each diagram element;
- show the resulting 3D scene.

To read a BPMN file the tool resorts to a parser, which takes as input a *.bpmn* file and produces a custom data structure.

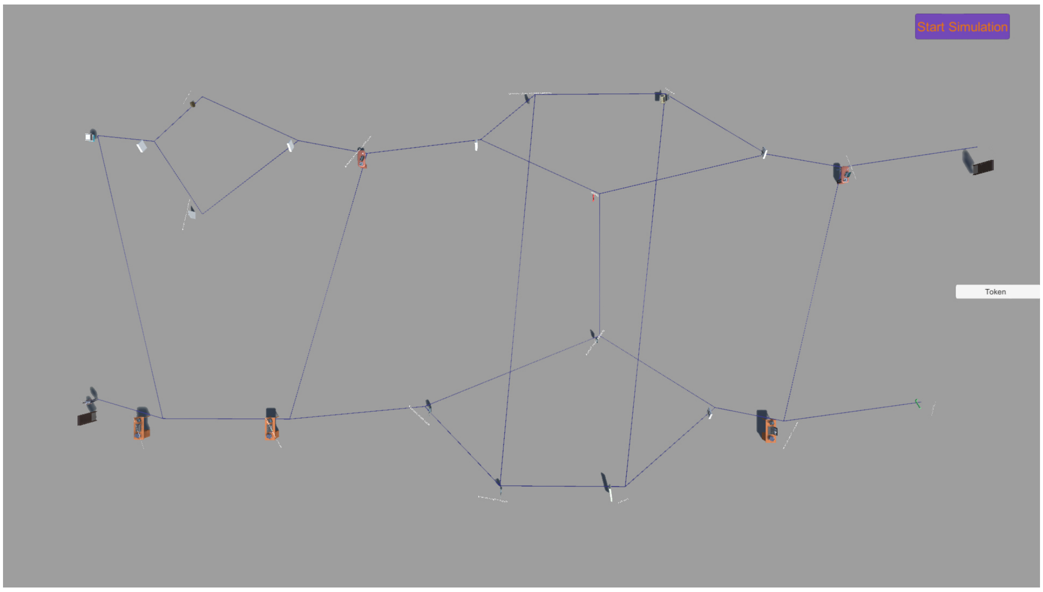


Figure 5.2: View from the top of the 3D world.

From the parsed model, UBBA collects the information regarding the elements contained in the diagram, such as the id, the name, the type (e.g., activity, gateway, event, etc.), the position in the 2D plane, and the list of nodes reachable from it. The tool exploits this information to set the 3D space. For each discovered element the tool allows to associate a 3D graphic; the designer can choose whether to load them from his/her PC or choose the ones already present in UBBA.

External graphics have to be *.fbx* files[2], available online for free or for sale. Even the tokens can be customized: the user can specify a personal graphic representing tokens generated by each pool in the diagram. The chosen graphics for elements and tokens are then embedded by the tool in a 3D space. Those graphics are positioned following the spatial information of the *.bpmn* file and connected by lines that represent sequence flows. Then a view from above of the model is shown, and the user can start to navigate it, see Figure 5.2.

The already created custom 3D world lets one introduce the animation feature of UBBA. It supports the diagram execution by means of token movements so that the user can continuously check their distribution. This facility results particularly useful for understanding not only the role of the token as process actor but also the semantics of the BPMN elements and the meaning of the diagram. The animation can be triggered by the user by clicking the *Play* button depicted on the top right side of the interface. In turn, from each start event the chosen 3D graphic, representing the token, starts to cross the diagram following the path indicated by sequence flows. The animation terminates once no more token moves are allowed.

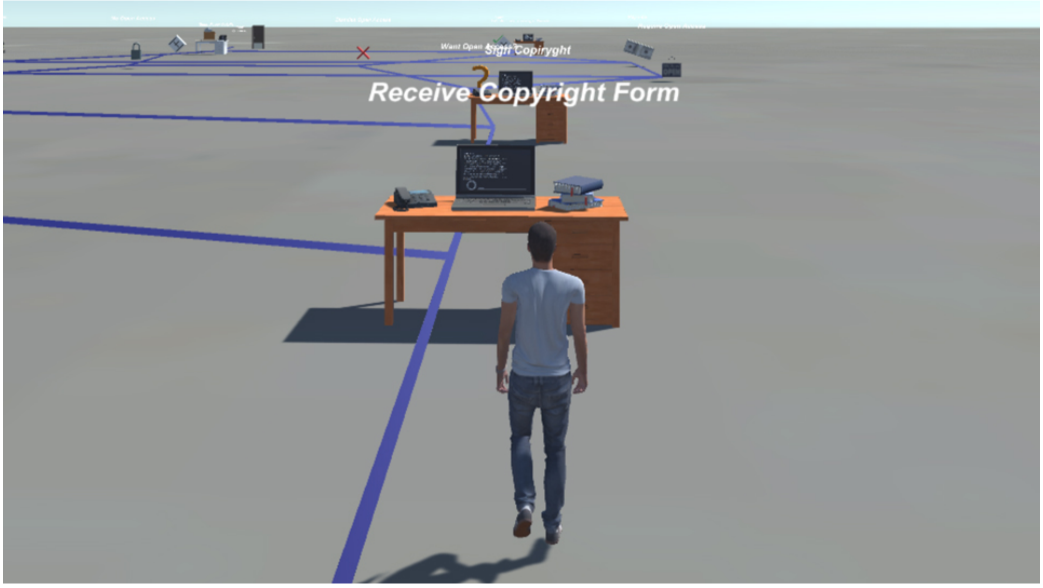


Figure 5.3: Token point of view.

[2] Filmbox (*.fbx*) is a file format for geometry definition widely adopted by 3D graphical application vendors, storing 2D, 3D, motion, audio, and video data.

A token can behave differently depending on the type of the node it is going to cross. UBBA implements the semantics of a core set of BPMN elements. Clearly, this collection is not the full list of BPMN elements, but it is enough to capture several semantic concepts.

More in detail, UBBA considers BPMN collaboration or process models with the following elements: *Task*, *Send Task*, *Receive Task*, *Exclusive Gateway*, *Parallel Gateway*, *Event-Based Gateway*, *Start Event*, *Start Message Event*, *End Event*, *Intermediate Catch Event*, and *Intermediate Throw Event*. The semantics of such elements is implemented in UBBA leaving the possibility of adding other element behavior just extending a project class.

During the animation, different points of views on the 3D environment are available for the user, who is free to switch from a camera to another one using the buttons on the right side of the interface. UBBA has a point of view for each active token, see Figure 5.3, plus another that covers the whole collaboration.

5.2 UBBA in Action

To show UBBA at work we present the implementation of a case study consisting in the BPMN collaboration diagram presented in Chapter 1, that is related to the scenario of the running example. For sake of presentation, Figure 5.4 repeats the diagram. The use here of a BPMN collaboration different from that used in the previous chapters is due to the need of a smaller diagram that permits to keep the focus more on the 3D animation.

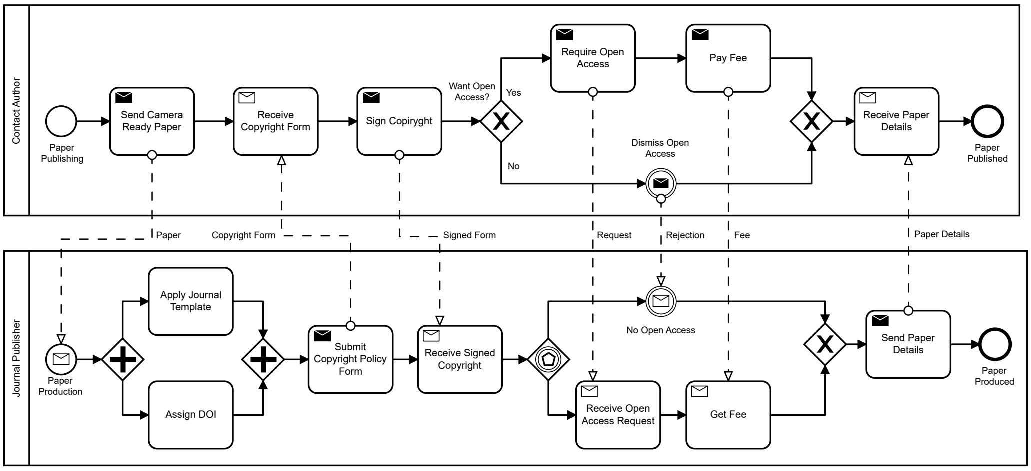
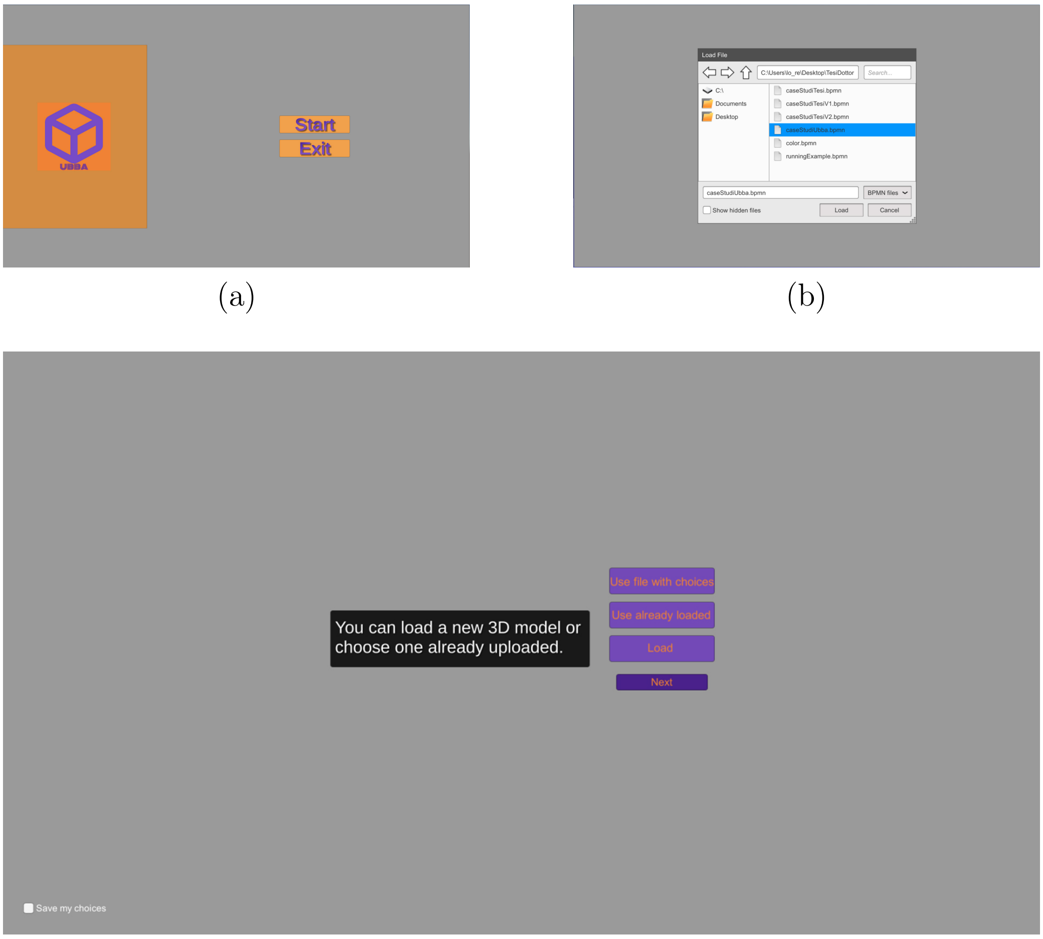


Figure 5.4: Paper publication collaboration.

For what concerns the UBBA functioning, it can be executed as a standard executable program. A double click on its executable file starts UBBA



C:\Users\lo_re\Desktop\TesiDottor

caseStudiUbba.bpmn

□ Show hidden files

You can load a new 3D model or choose one already uploaded.

□ Save my choices

(a) (b) (c)

Figure 5.5: UBBA startup.

and provides the first interaction interface, Figure 5.5 (a), and a file chooser where to load the collaboration diagram to animate. The tool comes with some example BPMN diagrams (e.g., the running example in Figure 5.4), but of course any *.bpmn* file can be chosen from the file system. Subsequently, UBBA needs element graphics to be associated to the collaboration elements. UBBA asks the user to select 3D from the tool assets or directly from the file system, see Figure 5.5 (c). Selecting the checkbox *Save my choices*, before moving on to the next step, the associations between BPMN elements and 3D graphics are saved in a preference file. Thus the next time the user want to animate the same model with the same 3D graphics, he/she can use the saved choices by pressing the *Use file with choices* button.

To recreate the setting described in the *Paper Publishing* collaboration diagram the designer needs to use graphics capable to embody together the element type and the specific meaning it has in the diagram. For instance, a mailbox is suitable for symbolizing a receive task in general, but in some cases it may results less effective if used in place of receiving tasks with specific meanings such as *Get Fee*. Indeed, the user has to consider the context of the business process in which the *Fee* message represents a money transfer, hence a 3D cash register or a wallet better clarifies this setting. The choice of the 3D graphics for the tokens is crucial, as well as a meaningful 3D graphic is essential to carry additional information on the diagram meaning. Tokens should represent the entities who perform the activities in the pool, see Figure 5.3. Alternatively, tokens describing more abstract actors, for instance, processes representing software components, can be depicted with a sphere. In the example, I associate two of *men* graphics to the tokens of *Journal Publisher* and *Contact Author* pools.

Once the association of the graphics is finished, UBBA generates the 3D model ready to be animated. Starting the animation, it is possible to appreciate the UBBA capability to represent the reality. The *man* graphics representing the author is placed near the *Paper Publishing* start event, which is represented by a 3D model of a door in order to mimic the entering in the process area. The *Contact Author* graphic crosses the door approaching the next element that is the *Send Camera Ready Paper* send task. It instead is rendered as it were the author's desk, where the PC notifies users by means of an envelope on the screen. The execution of this task involves the delivery of a message that UBBA depicts with a red sphere crossing the message flow, see Figure 5.6.

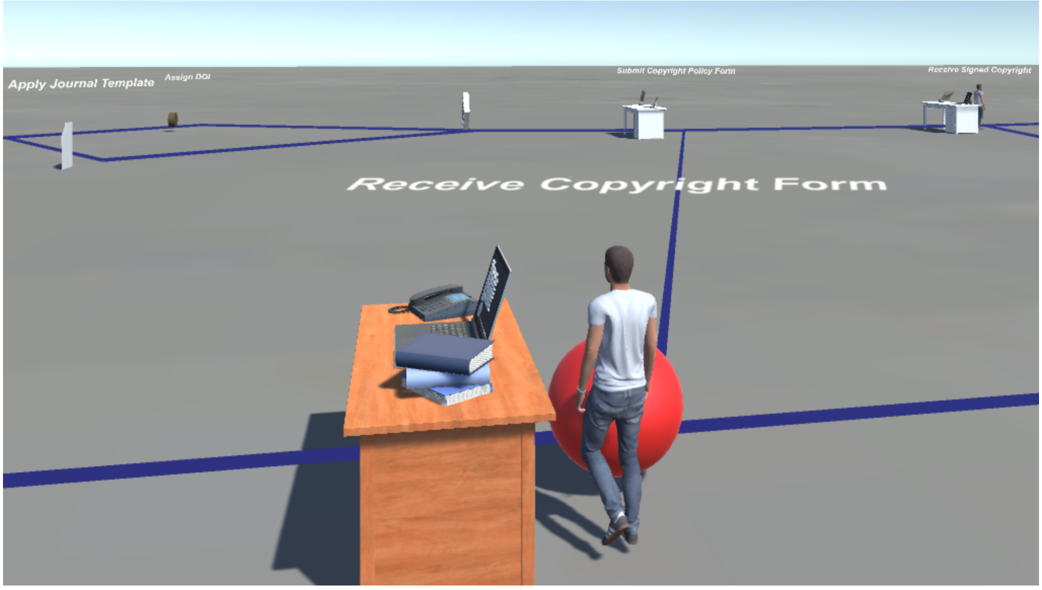


Figure 5.6: Message token.

The arrival of this message token triggers a new instance of the *Journal*

Publisher pool; having a new token in the collaboration, UBBA adds in the interface a new button identifying the *Journal Publisher* instance just created, see Figure 5.7. The button switches from the current view (i.e. whole collaboration or man perspective) to that of the publisher token.

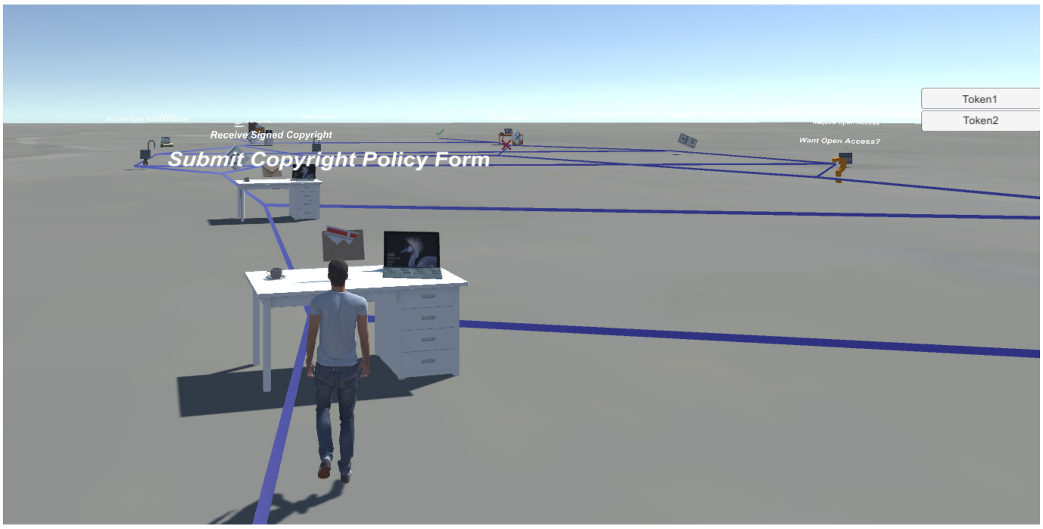


Figure 5.7: Camera selection.

Following the publisher token, it is possible to observe the processing of the received paper. In parallel, the publisher applies the journal template and the DOI to the paper. To render the parallel of the control flows, once it reaches a parallel split gateway, the token is cloned as many times as the number of parallel branches. While meeting a parallel join gateway, tokens are merged according to the operational semantics of the gateway. The collaboration continues with the publisher that sends to the author a copyright form to fill in. The author on its turn signs the form and gives it back to the publisher. This exchange of information is a block of two send and two receive tasks in the collaboration diagram, see Figure 5.4. UBBAuses another time graphics of desks; in this case the author and the publisher have different desks decorated with different accessories to explain the tasks' meaning.

Therefore, the author's token reaches the *Want Open Access?* exclusive split gateway, where he/she has to decide for an open access publication or not. The choice of the path to follow is interactive and made by the user; indeed he/she has to click on one of the 3D arrows depicted over the outgoing sequence flows, see Figure 5.9.

Following the *No* path, the *Contact Author* reaches a *red cross* graphic symbolizing the rejection. Otherwise, the *Yes* path brings the author to a graphic symbolizing the open access request. In both the cases, the author communicates its choice to the publisher through a message, but only in case

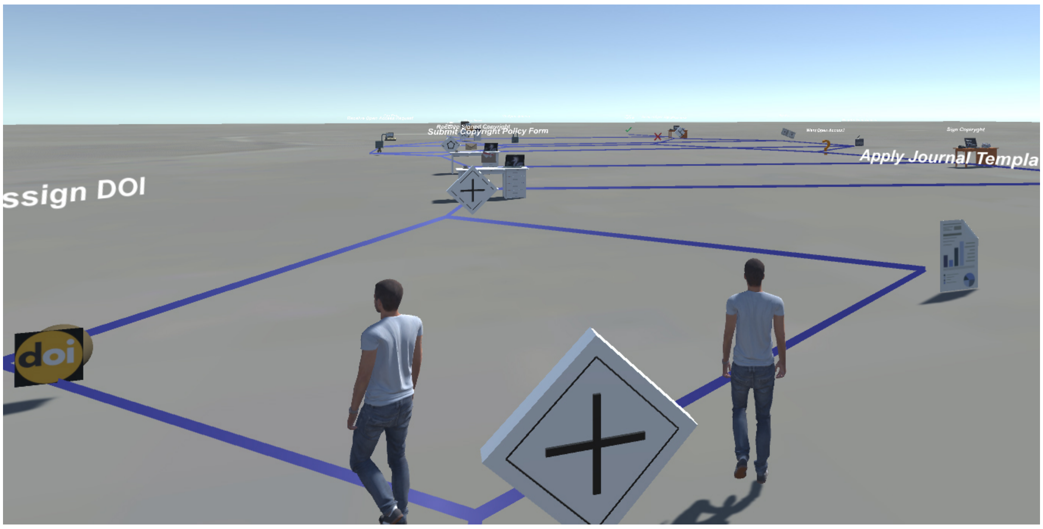


Figure 5.8: Parallel control flows.

the author requires the open access, he/she performs the *Pay Fee* send task. This regards a very straightforward activity that indeed is represented as a stack of cash.

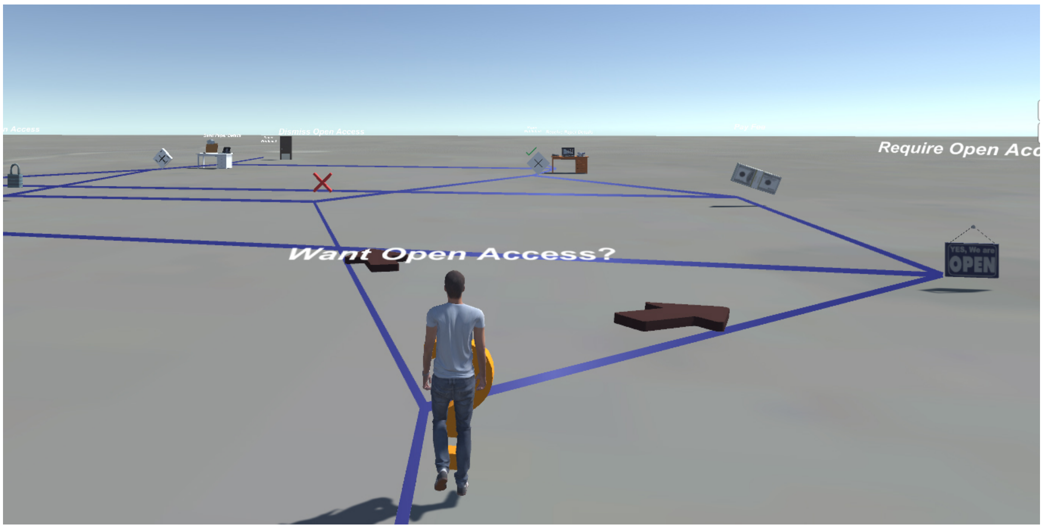


Figure 5.9: XOR splits in 3D.

Then the publisher reacts on the author's decision according to the semantics of the event-based gateway. An open access request brings the publisher token to an open lock graphics and to a cash register for collecting the *Fee*, while a rejection routes it to a closed one. To conclude the collaboration, the publisher sends to the author some details about the published paper. This is again rendered with desks, the one of the publisher has a PC on top showing the paper details, while the author's one has an envelope containing the same paper details.

5.3 Comparing with other Approaches

In recent years several works foster new techniques for modeling and visualizing organization processes capable to bridge the gap between business people and BPMN. On the one hand, a virtual world representing a business process can enhance the communication activities, thus facilitating interactions between businessmen and stakeholders [30]. On the other hand, the animation of business processes can increase their understanding [33, 24, 5] and also the possibility to debug them [17].

The literature proposes several tool prototypes that follow such principles. Indeed, in [9] an implementation of a 3D virtual BPMN editor that embeds process models into a 3D world is presented. Similarly, in [6] and [82] the representation of Petri net-based business process models is enriched with the third dimension. Virtual worlds have also been used in the context of Workflow Management Systems (WfMSs). In [31] the authors have implemented an agent-based simulation architecture that can be used as a simulation component for any WfMS. It is also worth mentioning the BPM simulation game *Innov8* by IBM [35]. It offers both IT and business players an excellent introduction to BPM, useful for learning the anatomy of a model. Another example is *Be a Token* [62], a JavaScript tool based on the A-Frame framework. This tool represents sequence flows as hallways to cross, and tasks as rooms with a door in the wall for each incoming/outgoing sequence flow. For what concerns the business process animation there are works attempting to show the processes execution, which however just provide a 2D visualization. These contributions use token flow or elements highlighting to indicate the current execution state of the process models. In [1], business processes are animated by means of a token game within the Signavio modeler, where users can step through the process element-by-element. Visual Paradigm [75] provides an animator that supports also collaboration diagrams. Finally, [20] provides an animator of BPMN collaborations enriched with data and multiple instances, which is based on token flow animation.

However, the above solutions suffer from three main limitations. Firstly, works recreating a 3D world do not provide any animation of the business process, but just visualization. This means that they statically show a 3D version of the model without supporting a representation of its execution. Moreover, these projects are not very customizable, but instead are limited to describing a particular setting without the possibility of using custom 3D models. Finally, works providing animation of business processes use only 2D environments.

Table 5.1 summarizes the features provided by the tools available in the literature and by UBBA. In detail it compares the tools with respect to their capability to: deal with BPMN *collaboration* models, *visualize* and *animate* in 2D or 3D the models' execution, insert *custom graphical elements*, and parse model files compliant with the *standard format*.

	[9]	[6]	[31]	[35]	[62]	[1]	[75]	[20]	UBBA
Collaboration	✗	✗	✗	✗	✗	✗	✓	✓	✓
Visualization	3D	3D	3D	▲	✗	2D	2D	2D	3D
Animation	✗	✗	✗	▲	3D	2D	2D	2D	3D
Custom Graphics	✗	✗	✗	✗	✗	✗	✗	▲	✓
Standard Input	✗	✓	✓	✗	✗	✓	✓	✓	✓

✓: fully, ▲: partially, ✗: not supported

Table 5.1: Literature comparison.

CHAPTER 6

PPLG: PURPOSE PARAMETRIC LOG GENERATOR

This chapter exploits the provided BPMN formal semantics for enabling process and collaboration diagrams simulation to automatically generate event logs suitable for a precise mining purpose. The following sections show the motivation behind the development of this log generation methodology. More in detail, we describe the methodology from a architectural point of view, and introduce a prototypical tool implementing the methodology through a use case example. Finally, we conclude with a tools comparison.

6.1 Motivations

We already mentioned the importance of process mining in the BP life-cycle, see Section 2.1. The role of process mining is also well recognized by companies, which appreciate the possibility to gather information from their processes [84]. In most of the cases, the effectiveness and the precision of mining techniques is strictly related to the reliability of the mining algorithms and/or the quality and the availability of event logs. When developing a process mining algorithm, it has to be tested against different event logs [22].

To evaluate the quality of a mining algorithm event logs are essential, especially those coming from real-world processes. In addition, to properly carry out a mining procedure, it may be necessary to refer to the *gold standard* [13, 60], i.e., the process models generating the log. However, it is quite unusual to find company logs directly extracted from IT systems, hence real processes. Indeed, organizations hardly spread their logs for analysis purposes, especially not coupled with the reference process diagrams [10].

In this regard, several approaches [10, 74, 22] in the literature started to investigate the automated generation of artificial event logs to overcome this issue. These approaches are used to execute several instances of the process models to generate the event logs. These instances behave differently according to the choices made by the simulator, that can be random or based on probability distributions. Therefore, the event logs made with these approaches cannot be tuned for specific purposes.

Depending on the *mining purpose* (e.g., rediscoverability, social network analysis, compliance checking, decision mining, what-if analysis), and on the applied mining algorithm, one may ask for event logs that guarantee particular properties [71]. For instance, the rediscoverability problem requires to guarantee that the event logs to analyze show all possible behavior of the process model, or at least all the directly following relations between activities. Using, for instance, the α -algorithm [70] in order to *rediscover* an exclusive choice like the one in Figure 6.1, the mined event log has to show traces where activity A is directly followed both by activity B and C.

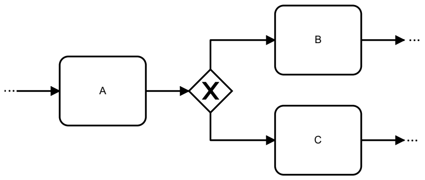


Figure 6.1: Rediscover an exclusive choice.

Another drawback regards the kind of information related to an event that a log generator can create. Indeed, the existing log generation tools simulate only the control-flow of the input process model. As consequence, most of the process mining techniques available today consider just control-flow related values to analyze business processes [45]. However, modern IT systems store huge amount of information that regards the execution of processes and collaborations. Apart from the attributes describing the control-flow (e.g., the activity name, the timestamp), event logs may carry resource, communication, and data related values.

To address these problems, we present a novel log generation methodology that is parametric on the input process model language and on the mining purpose, to produce artificial event logs. The methodology is meant to ensure the possibility of simulating any kind of process model through the implementation of several modeling language semantics (e.g., BPMN, Petri net, EPC, WF-net), and also the possibility to decide characteristics of the output event log according to the requirements of a mining procedure.

6.2 Event Logs Generation Methodology

This section presents the methodology, Figure 6.2, for generating automatically, from process models simulation, event logs suitable for the desired mining purpose. The methodology requires as input a business process model and a mining purpose, the result is an event log with specific properties directly related to that purpose.

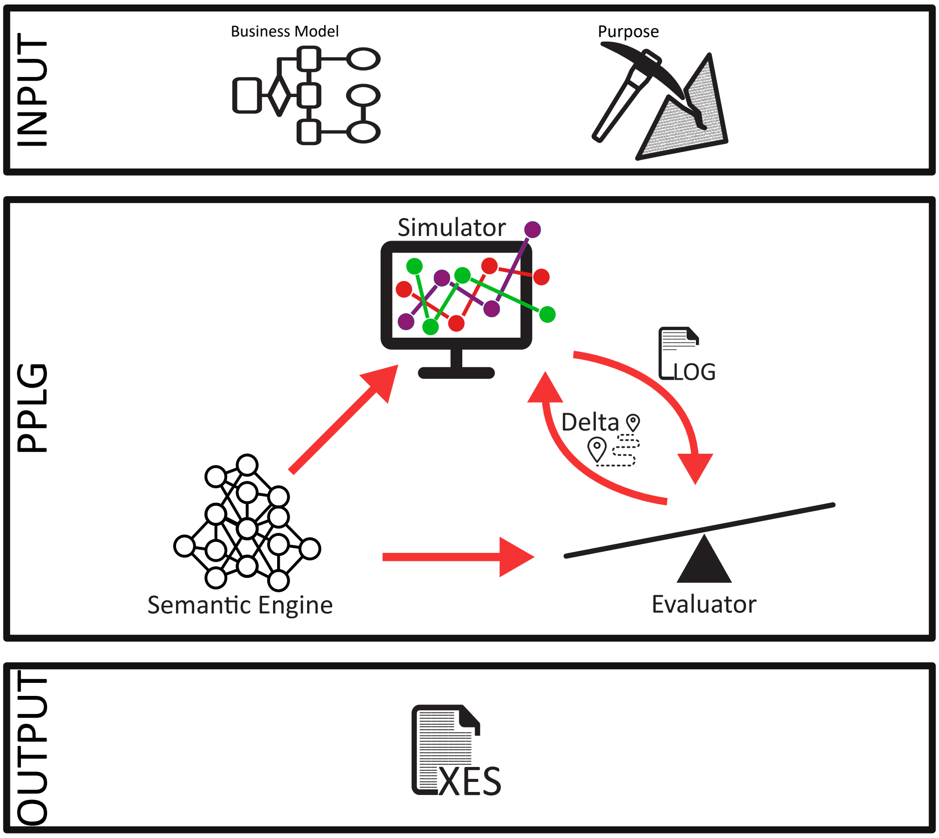


Figure 6.2: Purpose parametric log generation methodology (PPLG).

The generation of purpose aware logs is made possible through a *guided* simulation of the process model. The simulation is guided in the sense that it explores (and logs) just those traces needed for the mining purpose, rather than consuming resources in finding useless ones. Indeed, over the possibly infinite executions that a process model can perform, mining techniques may require only a subset of traces in order to accomplish their goal.

The duty of guiding the simulation is up to an evaluator that provides hints to the simulator. The evaluator is a function that compares the input model with the event log generated up to that moment by the simulator. It clearly depends on the mining purpose, in particular on the properties the final log must show. The result of the evaluation, called *delta*, represents the distance between the inputs of the comparison. More concretely, it contains parts of the execution traces that are missing in the event log. This *sub-traces* are the hints that act as bias in the next step of the simulation, guiding the simulator to execute traces containing them. Once the purpose is satisfied, i.e., the evaluation function returns a distance that is small enough for the user, the simulation terminates and the produced event log is given as output.

The methodology is independent from the modeling language of the business process given as input, and also from the mining purposes. It points at guiding the simulator in covering only the execution paths needed to satisfy the purpose. From an architectural point of view the methodology relies on the following components:

- the **semantic engine**; this component interprets the input process model and its semantics, it describes the possible executions of the process model;
- the guided **simulator**; this component has the duty of logging specific runs of the process model, it interacts with the semantic engine to know the possible executions, and receives hints from the evaluator to know the ones to log.
- the **evaluator**; this component compares the input model and the event log discovered through simulation and returns the resulting distance (i.e., *delta*) representing the hints for the next simulation run.

In the following some insights on the functioning of each of the components are given. Some of these components are described only in an abstract way through interfaces to be implemented. This permits to introduce additional features that guarantee the methodology to simulate several modeling languages for different mining purposes.

Semantic Engine. This component is responsible of parsing process models written in different notations.

```
public interface SemanticEngine {

    public Model parseModel(File input);

    public Map<Configuration, Set<Event>>
        getNexts(Configuration c);
}
```

Listing 6.1: Semantic Engine interface.

In this regard, different semantic engines should be implemented in order to enable the simulation of different languages. A new semantic engine can easily be produced following the contract of the semantic engine interface, Listing 6.1. It requires to implement a method for parsing the process model (line 3), and a method that serves the simulator to execute the process model (line 5).

The first method is straightforward, it reads an input file to produce a model data structure. The latter method implements the semantics of the modeling language. It provides the possible execution steps reachable from a process model configuration, and returns a set of reachable configurations and the related generated events. The semantic engine interface makes use of the same concept of configuration as in the formal semantics of Chapter 3, to represent a snapshot of the model execution.

Simulator. This component is the core of the methodology. Differently from the semantic engine, the simulator component does not require any customization to work in different settings.

```
1  public class Simulator {
2
3      private SemanticEngine engine;
4      private Log log;
5
6      public Simulator(SemanticEngine e) {
7          this.engine = e;
8          this.log = new Log();
9      }
10
11     @Override
12     public EventLog simulate(Delta delta) {
13         if (delta.isEmpty()) {
14             return randomSim(null);
15         }
16         for (Trace hint : delta) {
17             Set<Configuration> starts = LTS.find(hint.pop());
18             for (Configuration s : starts) {
19                 Trace prefix = getPrefix(s);
20                 log.append(guidedSim(prefix, s, hint));
21             }
22         }
23         return log;
24     }
25 }
```

Listing 6.2: Simulator implementation.

Listing 6.2 reports part of the simulator implementation: the constructor that initializes the simulation on the basis of a semantic engine, and the *simulate* method at line 12. This method runs through the semantic engine new model executions based on the input parameter *delta*. *Delta* is the product of the evaluation that guides the simulator, it contains sub-sequences of execution traces. It is empty at the first run of simulation, because the evaluator has not performed any comparison yet. The empty *delta* makes the simulation execute the model in a random way (line 14), logging one of the possible execution traces of the input diagram. Otherwise, the simulation loops the hints contained in the *delta* (line 16). These hints are lists of events that the simulator uses to guide the simulation and to log new traces. To achieve this goal, the simulator uses a partial labeled transition system of the diagram, generated by previous simulation runs. It serves to find one or more possible configurations that correspond to the execution of the first event in the considered hint (line 17). For each of these configurations, the simulator creates a prefix trace that leads to this configuration (line 19), and calls the *giudedSim()* method in order to complete them with events that include the remaining part of the hint.

```
1 private Trace guidedSim(Trace prefix, Configuration current,
      Trace hint){
2     if (hint.isEmpty()) return prefix;
3     Configuration next = hint.pop();
4     Path path = findPath(current, next);
5     if(path.isEmpty()) return randomSim(prefix);
6     prefix.add(path.getEvents());
7     retrun guidedSim(prefix, next, hint);
8 }
```

Listing 6.3: Giuded simulation.

Indeed the guided simulation method, Listing 6.3, takes as input the prefix, the current configuration, and the remaining part of the hint (the head element of hint has been retrieved at line 17 of Listing 6.2). The method is recursive, it tries to find a path connecting the current configuration to the next defined in the hint. Then, in the recursive case, the method adds the discovered path to the prefix and calls itself with the new parameters (line 7). In case of an empty hint or path, the base cases (lines 2 and 5) are activated. An empty path means that the simulator cannot provide the desired trace (so it returns a random one), the empty hint, instead, means that the entire trace has been found.

Evaluator. Finally, there is the evaluator. This component is devoted to optimize the work of the simulation. After any simulation run, the evaluator receives the current event log. This has to be compared with the business diagram by means of an evaluation function that depends on the mining purpose. To permit to extend the framework, the evaluator interface exposes the method *evaluate* to be implemented, Listing 6.4.

```java
public interface Evaluator {

    public Delta evaluate(EventLog discovered, SemanticEngine
        engine, Integer tau);

}
```

Listing 6.4: Evaluator interface.

The implementation of the evaluation is not an easy task. Indeed, it has to compare objects of different categories: an event log and a diagram. However, in terms of process executions, these concepts can be compared. Indeed, an event log represents just a subset of the possible diagram executions. The same for the event log to achieve, the goal of the purpose, and for *delta*. All these concepts can be represented as parts of a transition system, like in Figure 6.3, the gray area (possibly infinite) represents the possible executions of the model, that include all the subsets, in red the discovered log, in green the goal of the purpose, and *delta* in purple.

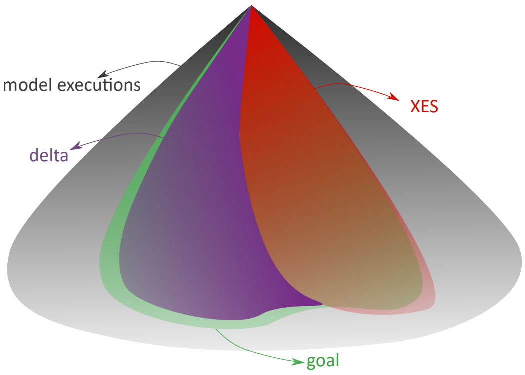


Figure 6.3: Delta evaluation.

Therefore, to implement a new evaluation function for a mining purpose one needs to transform the members of the comparison onto comparable entities, the resulting difference will specify the *delta*. An evaluation result lower than a tolerance (tau) value set in input concludes the generation of the log. Indeed, it means that no more traces are required in the produced event log.

6.3 Rediscoverability Example in PPLG

This section introduces the prototypical tool PPLG (Purpose Parametric Log Generator) that implements the described methodology, and presents a use case example in order to better explain how each component of the methodology works. Finally, it concludes comparing PPLG with other log generator tools.

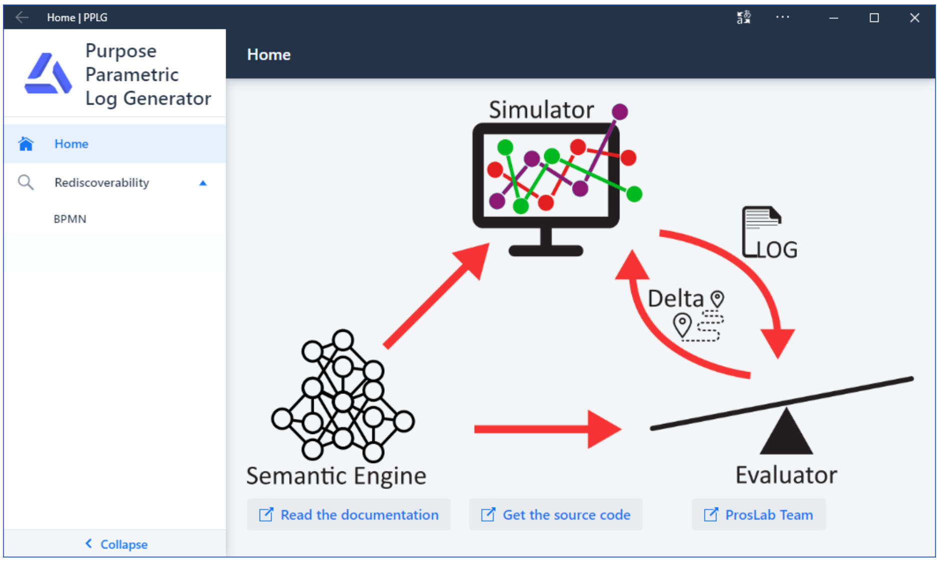


Figure 6.4: Graphical interface of PPLG.

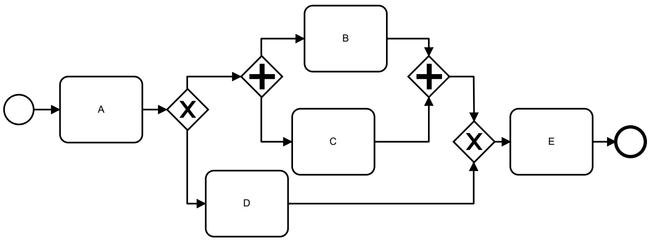
PPLG, Figure 6.4, is an extensible progressive web app written in *Java* using the full stack developing framework *Vaadin*[1]. Thus, it can be executed in any operating system. The tool is still under development, but it can be already downloaded and used. The source code both with usage instructions are available at `https://bitbucket.org/proslabteam/guidedsimulator/`, while builds of PPLG are available at `https://bitbucket.org/proslabteam/guidedsimulator-builds/src/0.1/`.

The tool already implements the rediscoverability purpose, through the evaluation of the ordering relations between tasks, for the BPMN language. PPLG implements the BPMN semantics described in Chapter 3 and its relaxed version which discards data related features. Thus, it can read and simulate BPMN processes and collaborations with multiple instances, data and messages, to produce multi-perspective event logs.

To better show the methodology and the tool functionalities, we present in the following an example concerning the generation of an artificial event log from the BPMN process model in Figure 6.5 (a). The purpose is the rediscoverability by means of the α-algorithm. This algorithm aims at recreating the Petri net from the event log it has generated.

In a nutshell, the α-algorithm analyzes the sequences of events composing the log to find the order relations between tasks (e.g., if two tasks act in

[1] `https://vaadin.com`



(a)

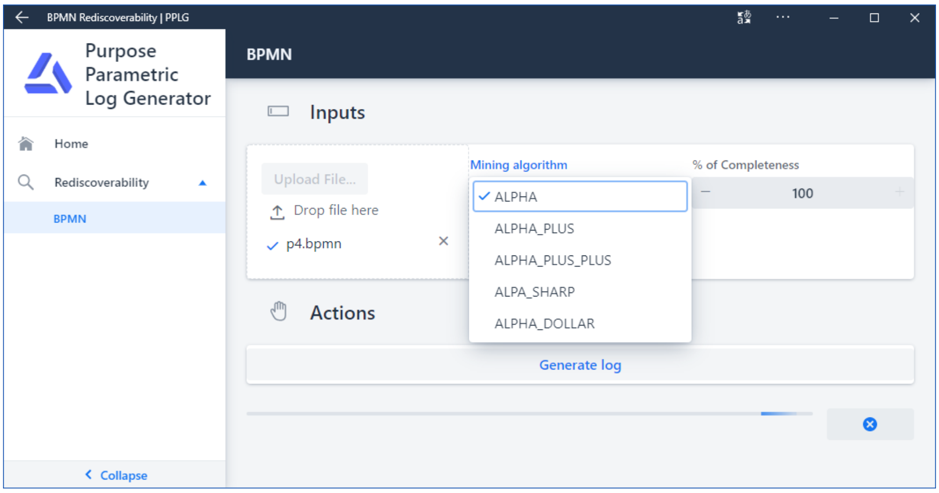
	A	B	C	D	E
A	#	→	→	→	#
B	←	#	||	#	→
C	←	||	#	#	→
D	←	#	#	#	→
E	#	←	←	←	#

→: sequential, ||: parallel,
#: exclusive

(b)

Figure 6.5: The input process model (a), and the related order relations matrix (b).

parallel, or one after the other). These dependencies come from the order relations between tasks [70, Definition 3.2] discovered in the log, and drive the algorithm during the process reproduction. Therefore, the result of the algorithm directly depends on the log quality, having, for instance, multiple repetitions of the same trace is useless. Considering the example, the matrix in Figure 6.5 (b) contains the order relations of the input process diagram (e.g., cell (1,2) indicates that task A always precedes task B, $A \rightarrow B$). Thus, the simulation has to be guided, by means of *delta*, into the discovery of those relations.



✓ p4.bpmn

Mining algorithm: ✓ ALPHA

% of Completeness: 100

Figure 6.6: Available rediscoverability algorithms in PPLG.

The example is carried out step by step with the help of PPLG that already implements the rediscoverability purpose, Figure 6.6, for a class of mining algorithms that exploit order relation between tasks, like the α-algorithm. Moreover, PPLG is integrated with external software for applying the α-algorithm (i.e., ProM) and converting Petri nets into BPMN diagrams.

PPLG performs a looping routine, see Figure 6.7, that terminates once the produced log acquires the desired characteristics. In the following, we show each step performed by the tool to reach the goal imposed in the example.

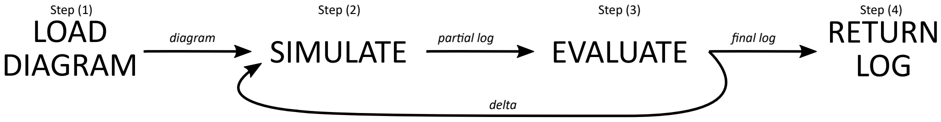


Figure 6.7: PPLG routine.

Step (1) consists in load of the input diagram and choose the mining algorithm to use in the semantic engine. This creates a data structure containing the diagram information, and provides to the simulator the method for executing it. At this stage, PPLG is ready to simulate the model being aware of the hints given by the evaluator, thus it starts to loop between steps (2) and (3).

Step (2) invokes the simulator that does not receive any hint (i.e., an empty *delta*). This leads to perform a random execution of the process model. Listing 6.5 shows the trace resulting from the first simulation run of the prototypical tool. In this case, the simulator performed tasks A, B, C and E, one after the other.

```
<trace>
   <string key="concept:name" value="case_1"/>
   <event>
      <string key="concept:name" value="A"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.825+01:00"/>
   </event>
   <event>
      <string key="concept:name" value="B"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.850+01:00"/>
   </event>
   <event>
      <string key="concept:name" value="C"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.856+01:00"/>
   </event>
   <event>
      <string key="concept:name" value="E"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.886+01:00"/>
   </event>
</trace>
```

Listing 6.5: First random trace.

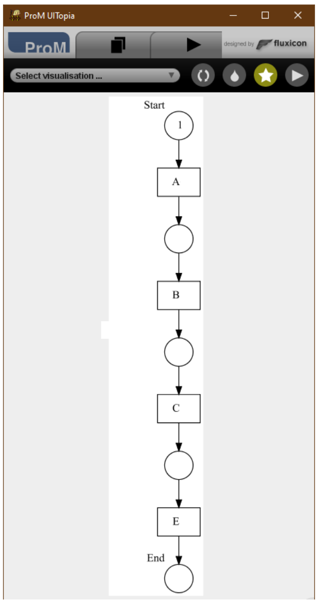


Figure 6.8: First resulting Petri net.

Step (3) is the evaluation of the current log according to the mining purpose that is the discovery of a process model behaviorally equivalent to the original one. This requires mining the current log to calculate its order relation matrix. The mining is done by means of the ProM tool implementation of the α-algorithm. Being present just one trace in the log Listing 6.5, the result of the mining in ProM is straightforward, and it is the Petri net depicted in Figure 6.8. Thus, the Petri net is transformed into a BPMN diagram using the converter of ProM that follows the mapping described in [25]. The order relations of the resulting model are calculated by PPLG, and compared with the ones of the original process model. All those relations that are missing in the current mined log identify how far the tool is from its goal. Moreover, the missing relations become the *delta* for the next simulation steps.

In the example, the first round of simulation identifies three order relations that are $A \rightarrow B$, $B \rightarrow C$, and $C \rightarrow E$. The evaluator compares them with the ones of Figure 6.5 (b), getting the missing ones: $A \rightarrow C$, $A \rightarrow D$, $B \rightarrow E$, $D \rightarrow E$, and $C \rightarrow B$. These missing relations are then translated into the *sub-traces* composing the *delta*.

Step (2) is performed again. The simulator gets a *delta* of this form: $\{\langle A, C\rangle, \langle A, D\rangle, \langle B, E\rangle, \langle D, E\rangle, \langle C, B\rangle\}$. *Delta* is crucial to drive the simulator during the search for additional traces containing the missing relations. At this point, the simulator exploits the *delta* and the knowledge acquired in the previous run. Indeed, it already discovered one of the possible process executions. Figure 6.9 shows the labeled transition system generated by the process model. The green states are the goal of the simulation, as they generate the order relations. The ones filled in red are instead those states previously discovered by the simulator. Finally, the violet and thicker transitions represent the *delta*, the guide for the new simulation run.

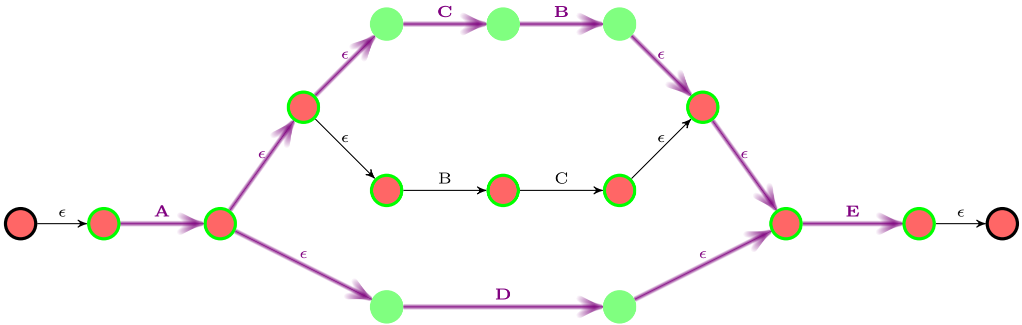


Figure 6.9: Discovered process behavior.

Starting from the already known states in the transition system, the simulator executes new traces following the transitions highlighted by *delta*. This run produces the new traces $\langle A, C, B, E\rangle$ and $\langle A, D, E\rangle$, as showed in Listing 6.6. These trace are finally joined with the one previously discovered, and the entire log is passed again to the evaluator.

Step (3) again requires to mine the received log. The resulting Petri net is the one in Figure 6.10, and its order relations matrix in this case, is exactly the same as the one generated with the original process model. Therefore, the tool can terminate the looping routine.

```
<trace>
   <string key="concept:name" value="case_2"/>
   <event>
      <string key="concept:name" value="A"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.887+01:00"/>
   </event>
   <event>
      <string key="concept:name" value="C"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.903+01:00"/>
   </event>
   <event>
      <string key="concept:name" value="B"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.905+01:00"/>
   </event>
   <event>
      <string key="concept:name" value="E"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.913+01:00"/>
   </event>
</trace>
<trace>
   <string key="concept:name" value="case_3"/>
   <event>
      <string key="concept:name" value="A"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.943+01:00"/>
   </event>
   <event>
      <string key="concept:name" value="D"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.950+01:00"/>
   </event>
   <event>
      <string key="concept:name" value="E"/>
      <date key="time:timestamp"
           value="2020-02-20T16:54:30.957+01:00"/>
   </event>
</trace>
```

Listing 6.6: Second random trace.

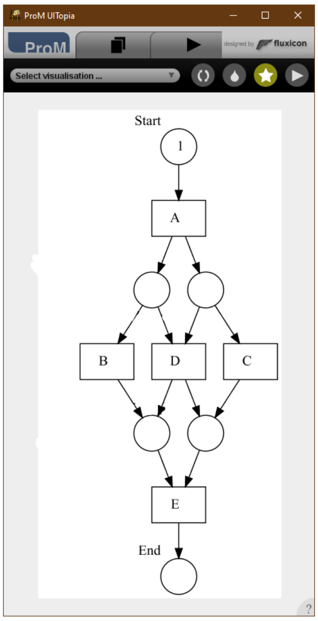


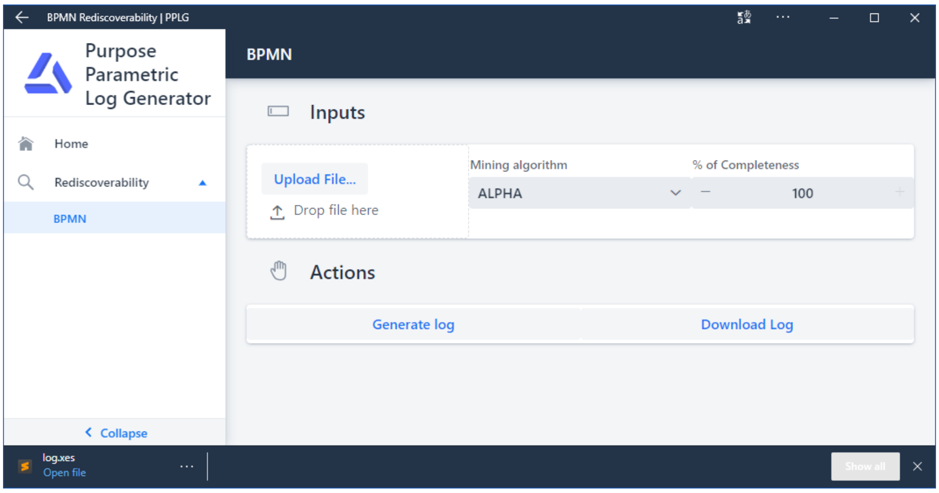
Figure 6.10: Final resulting Petri net.

Step (4) produces as output the *.xes* file, Figure 6.11 containing the simulated log. It contains the only traces, with no repetition, required by the purpose.

To better highlight the advantage of the methodology, we presents a comparison testing the ridescoverability purpose in PPLG against other log generators (i.e., PLG2[2] and BIMP[3]). We took a set of ten BPMN process models used as input, half of the diagrams are artificial process models generated by PLG2, while the others comes from the literature. The comparison consists in checking the quality of the event logs generated by the three tools with respect to the activity relations matrix of the input process model. More in detail, we calculate the percentage of activity relations in the matrix that are also present in the log.

[2] `https://plg.processmining.it/`

[3] `https://bimp.cs.ut.ee/`



Mining algorithm: ALPHA

% of Completeness: 100

Figure 6.11: Event log serialization.

Being that the generation of logs in PLG2 and BIMP does not depend on any purpose, these tools ask for a number of traces to simulate. Differently, PPLG autonomously evaluates whether the produced log satisfies the purpose and stops the simulation. Thus, we firstly ran the generation of logs with PPLG in order to fix the number of traces to be simulated.

Then, we ran the simulation also in PLG2 and BIMP asking for the same number of traces. As result, for each of the tested process model we obtained three event logs, one from each tool, and we compared them with respect to the relations matrix.

Model	**Elements**	**Traces**	**PLG2**	**BIMP**	**PPLG**
p0	10	3	100%	66%	100%
p1	11	3	66%	66%	100%
p2	12	5	100%	83%	100%
p3	17	5	90%	90%	100%
p4	21	10	62%	44%	100%
p5	27	10	95%	79%	100%
p6	34	14	68%	52%	100%
p7	40	76	66%	37%	100%
p8	49	226	14%	7%	100%
p9	53	41	51%	41%	100%

Table 6.1: Rediscoverability results.

Table 6.1 resumes the results of this comparison. The first two columns, *Model* and *Elements*, contain respectively the name of the process model and the number of its elements. Column *Traces* provides the number of traces generated by PPLG to cover all the activity relations and used as reference for the other tools. The last three columns show the percentage of activity relations covered respectively by PPLG, BIMP, and PPLG.

Being guided by the evaluator, PPLG covered entirely the relations matrix for each of the tested process models. While the other tools show worse results especially in case of bigger models. Consequently, applying the α-algorithm on the event logs generated by PLG2 and BIMP will produce models very different from the original, while the logs of PPLG can produce exactly the same model.

6.4 Comparing with other Approaches

The literature provides several works concerning the generation of artificial event logs through the simulation of business models. Alves and Günter [22] tackle the generation of event logs from CP-nets [37]. This work highlights the issues that may arise when using real-life event logs to fine tune mining algorithms. The presence of noise or the incompleteness of an event log can indeed compromise the evaluation of an algorithm. The solution proposed by the authors is developed on top of the CPN tool [38] for what concerns the model simulation, and is coupled with the ProM framework for generating the logs. The major drawbacks of this work regard the generated logs themselves. Their generation cannot be tuned to produce different results according to the usage. Moreover this work uses the *MXML* [72] format for exporting the logs, because it precedes the development of the *XES* standard, causing problems of incompatibility with several mining algorithms.

Burattin overcomes this limitation proposing in [10] a tool called Process Log Generator (PLG2). The tool allows to generate event logs from the simulation of BPMN processes according to desired characteristics, or loaded by the user. PLG2 simulates BPMN processes with a core set of elements (e.g., tasks, parallel/exclusive gateways), and produces event logs in the *XES* format. This work considers data objects, that can store variables (and their relative values). During the simulation, tasks linked to a data object produce an event log enriched with the data object variable. Moreover, PLG2 considers time elapsed for executing tasks that can be set by the user by means of a script. Hence, the logs present two events (begin and end) for the same activity.

Similarly, Kataeva and Kalenkova present in [39] a tool for generating just well-structured BPMN diagrams through context-free grammars. The models can be simulated to generate event logs.

Mitsyuk et al. face in [50] the problem of defining and generating logs from collaborative processes including data flow and time. They formalize an executable BPMN semantics by which they simulate collaborative models. The proposed semantics support a subset of the BPMN elements such as tasks (also send/receive), sub-processes, parallel and exclusive gateways and cancellation events. Data objects store single data values used for driving exclusive choices, anyway messages cannot carry data. The simulation produces event logs for training mining algorithms. It is noteworthy that in case of multiple actions enabled in the model, the proposed simulator firstly enacts gateways and then randomly chooses a task. This is to simulate the atomicity of the gateways execution. Anyway, tasks are chosen randomly from among the different processes, each of them having the same probability to be run.

Stocker and Accorsi introduce in [63] an approach for generating event logs, focusing on a precise business scenario. Indeed, they present a simulator called *SecSy* to generate logs for training the modeling of security oriented systems.

Summing up, differently form the presented methodology, these works mainly focus on generating event logs without a specific mining purpose. Moreover, there is lack in managing other model languages and intricate aspects, such as multiple instances, data, and messages.

	[22]	[10]	[39]	[50]	[63]	Us
Input language	CPN	BPMN	BPMN	BPMN	BPMN	any
Collaborations	✗	✗	✗	✓	✗	✓
Multi-instance	✗	✗	✗	✗	✗	✓
Data	✗	✓	✗	✓	✓	✓
Output format	MXML	XES	XES	XES	XES	XES
Purpose aware	✗	✗	✗	✗	✗	✓

✓: fully, ✗: not supported

Table 6.2: Literature comparison.

In table 6.2 we summarize the key features of the tools presented in the literature review and the ones expected in our prototypical tool. In detail, we compare the the following capabilities: the *language* of the input model; the possibility to manage *collaborative*, *multi-instance*, and *data-aware* models; the *file format* of the generated log; and the ability to tune the simulation due a specific mining *purpose*.

PART IV

CONCLUSIONS AND FUTURE WORK

CHAPTER 7

CONCLUSION AND FUTURE WORK

The goal of the research presented in this thesis is twofold. On the one hand, it formalizes an extensive part of the BPMN notation; on the other hand, it exploits this formal semantics for the development of animation and simulation tools. This thesis presents in Chapter 3 a direct formal semantics in the operational style for an extensive set of BPMN elements. Being close to the standard, the semantics enables to catch the language peculiarities such as multiple instances, communication, and data. The formalization is supported by a running example. In particular, it makes use of the *Reviewing Procedure* collaboration to show the textual notation, and explain the operational semantics.

Beside this formalization, it is also made possible to animate collaborations. Chapter 4 introduces MIDA, an animator that turns out to be an effective supporting tool for *enhancing the understanding* of BPMN collaborations, and *debugging* errors that can easily arise when modeling them. Chapter 5 furthermore exploits animation through UBBA. It enables the representation, as well the animation, of collaboration diagrams as 3D virtual worlds. The proposed BPMN semantics is partially implemented also in UBBA to guarantee a proper animation of the model. Chapter 6 provides instead use of the formal semantics in the field of process mining. It presents a log generation methodology parametric to the model's language and to the mining purpose.

The following sections conclude the thesis by discussing results of the work, and the assumptions and limitations of the given approach, also touching upon directions for future work.

7.1 Thesis Results

This thesis contributes to the definition of a direct BPMN operational semantics, providing a uniform formal framework to exploit for animation and simulation purposes. The operational semantics clarify the interplay between control features, data, message exchanges and multiple instances in the BPMN models. The framework can manage BPMN elements with complex semantics like the OR-Join gateway and the multi-instance elements. In particular, besides multi-instance pools, the semantics support multi-instance tasks with different execution modalities, resulting from the combination of parameters concerning atomicity, concurrency and sequentialization/parallelization of task instances. Moreover, it includes all kinds of data elements provided by BPMN: *data objects* and *data collections*, which are local to process instances, and *data stores*, persistently storing data shared among different instances.

Besides being useful *per se*, as it provides a precise understanding of ambiguous and loose points of the standard, a main benefit of this formalization is that it paves the way for the development of supporting tools. Indeed, animation tools based on the given formal semantics are proposed. They provide the visualization of the behavior of a collaboration both in 2D and 3D. It is indeed well recognized that process animators play an important role in enhancing the understanding of business processes behavior and that, to this aim, a faithful correspondence with the semantics is essential, although this is not always supported.

In detail, through a running example, it has been shown that MIDA supports model designers in achieving a priori knowledge of collaboration behavior, in terms of executed activities, exchanged messages, and evolution of data values for each active instance. This allows designers to *debug* their collaboration models. In this way, they can detect, and hence prevent, undesired executions, where, e.g., a control flow is blocked or an erroneous interaction arises. Designers can deduce the cause beyond them by checking the evolution of token distribution and of involved data. MIDA animation features result helpful both in educational contexts, for explaining the behavior of BPMN elements, and in practical modeling activity. Notably, in addition to the *Paper Reviewing* collaboration, the MIDA web page makes available a collection of BPMN models referring to different scenarios ready to be animated (e.g., travel booking, smart home heating system management, procedures for student internship and exam registration)[1]. Each example is provided in different variants, to show to the user how MIDA can spot different execution issues. UBBA fosters the visualization of collaboration

[1] These BPMN models are available at: `https://bitbucket.org/proslabteam/mida/src/36a18b195b5a/assets/examples/`.

diagrams moving the animation into 3D virtual worlds. It results in an improved communication between business analysts and stakeholders. Indeed, UBBA can effectively help people in sharing business knowledge. In addition to animation, the provided operational semantics permits to simulate BPMN diagrams. This technique has been widely exploited to generate artificial event logs just for a generic mining purpose. The methodology, and the prototypical tool PPLG presented in this thesis push forward this limit, being parametric in the modeling language and in the mining purpose.

7.2 Assumptions and Limitations

For what concerns the formal semantics, it focuses on the communication mechanisms of collaborative systems, where multiple participants cooperate and share information. Thus, we left out those features of BPMN whose formal treatment is orthogonal to the addressed problem, such as timed events and error handling. Moreover, to keep the proposed formalization more manageable, sub-processes are left out as well, despite the fact that they can be relevant for multi-instance collaborations. Introducing sub-processes in the formalization cannot be done by including it as a mere macro. In fact, in general, simply flattening a process by replacing its sub-process elements by their expanded processes results in a model with different behavior. This is because a sub-process, for example, delimits the scope of the enclosed data objects and confines the effect of termination events. Therefore, it would be necessary to explicitly deal with the resulting multi-layer perspective, which adds complexity to the formal treatment. The formalization would become even more complex if considering multi-instance sub-processes, which would require an extension of the correlation mechanism.

Considering the animation, MIDA has a good level of maturity, it has been used from 7 countries all over the world to animate more than 100 models per month. Its main limitation is the development platform. Indeed, even if Javascript applications are light and flexible, they are not suitable for tasks with heavy calculations like simulation. Differently, UBBA is a more recent tool that shows some defects. It focuses on just a core set of BPMN elements that is clearly less expressive than the one of MIDA. However, the rendering of several concepts in 3D is not so easy (e.g., multiple instances and data).

About simulation finally, the level of maturity of the log generation methodology is still growing, the work is currently under development and needs validation. For what concerns PPLG, it presents several components already implemented that permit to instantiate the rediscoverability purpose. However, other purposes and additional modelling languages are still missing.

7.3 Future Work

We plan to continue this work to effectively support modeling, animation, and simulation of BPMN multi-instance collaborations, by overcoming the above limitations. In particular, we intend to extend the treatment of the data perspective, by considering more sophisticated definitions of the internal structure of data elements (e.g., based on UML [66]) with respect to the generic record structure considered here. In addition, we plan to investigate other expression languages operating on data (e.g., the language FEEL [54, Ch. 10]). Moreover, we plan to extend the work, both from the formal and practical perspective. To this aim, we intend to first define a symbolic formal semantics, like that in [7], and then to extend the implementation of the semantics accordingly. The use of SMT solvers will be considered to deal with the constraints generated by the symbolic semantics. We also plan to exploit the formal semantics, and its implementation, to enable the verification of properties using, e.g., model checking techniques.

About MIDA, we plan to add new functionalities to further help designers in debugging their models. In this regard, we would like to decouple the semantics from MIDA, and implement it server side (exploiting the Java implementation showed in Chapter 6). This would enable the possibility to perform tasks heavier than animation, like formal verification of behavioral properties (e.g., soundness, safeness [19]). Moreover, we also want to foster the modeling and the animation introducing new features. One could be backward animation, to track back model execution. This, in conjunction with the possibility to make run-time changes on the model should speed up model debugging.

With respect to UBBA, several further developments can be carried out. By now, we plan to increase the core set of BPMN elements number and the set of possible customizations, allowing the user to associate 3D graphics also to sequence/message flows, message tokens, ans pools. Finally, in order to assess the potential and the scalability of the approach, We plan to conduct a validation with groups of business process designers, composed by: students with an academic knowledge of BPMN; designers from industry that have more practical skills; and stakeholders that are domain expert. The validation should provide feedback both on the usability of UBBA and on the quality and the benefits of the produced 3D animation.

Considering the log generation methodology, there are plenty of possible further developments. Firstly, we plan to finalize the development of the prototypical tool, and to define new case studies for different mining purposes to get insights on the validity of the methodology. Moreover, to foster the prototype extension, we will use wrappers to integrate external tools. For instance, we would like to integrate formal environments, like Maude [14], to easily increment the number of managed modeling languages, but also process mining tools, like Disco [29], to help the definition of new evaluation functions, hence the handling of new mining purposes.

BIBLIOGRAPHY

[1] Thomas Allweyer and Stefan Schweitzer. A tool for animating BPMN token flow. In *BPMN Workshop*, volume 125 of *LNBIP*, pages 98–106. Springer, 2012.

[2] Banu Aysolmaz. PRIME Process Animation. `http://prime.cs.vu.nl/`.

[3] Banu Aysolmaz and Hajo Reijers. Towards an integrated framework for invigorating process models: A research agenda. In *Business Process Management Workshops*, volume 256 of *LNBIP*, pages 552–558. Springer, 2016.

[4] Jörg Becker, Martin Kugeler, and Michael Rosemann. *Process management: a guide for the design of business processes*. Springer Science & Business Media, 2013.

[5] Jörg Becker, Martin Kugeler, and Michael Rosemann. *Process management: a guide for the design of business processes*. Springer Science & Business Media, 2013.

[6] Stefanie Betz, Daniel Eichhorn, Susan Hickl, Stefan Klink, Agnes Koschmider, Yu Li, Andreas Oberweis, and Ralf Trunko. 3D Representation of Business Process Models. *MobIS*, 8:73–87, 2008.

[7] Michele Boreale and Rocco De Nicola. A symbolic semantics for the pi-calculus. *Inf. Comput.*, 126(1):34–52, 1996.

[8] Egon Börger and Bernhard Thalheim. A Method for Verifiable and Validatable Business Process Modeling. In *Advances in Software Engineering*, volume 5316 of *LNCS*, pages 59–115. Springer, 2008.

[9] Ross Brown. Conceptual Modelling in 3D Virtual Worlds for Process Communication. In *Asia-Pacific Conference on Conceptual Modelling*, volume 110, pages 25–32. Australian Computer Society, Inc., 2010.

[10] Andrea Burattin. PLG2: multiperspective process randomization with online and offline simulations. In *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2016*, volume 1789 of *CEUR Workshop Proceedings*, pages 1–6. CEUR-WS.org, 2016.

[11] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. Foundations of Data-aware Process Analysis: A Database Theory Perspective. In *Proceedings of the 32td ACM Symposium on Principles of Database Systems*, pages 1–12. ACM, 2013.

[12] David Raymond Christiansen, Marco Carbone, and Thomas Hildebrandt. Formal Semantics and Implementation of BPMN 2.0 Inclusive Gateways. In *Web Services and Formal Methods*, volume 6551 of *LNCS*, pages 146–160. Springer, 2011.

[13] Krzysztof Cios, Witold Pedrycz, Roman Swiniarski, and Lukasz Andrzej Kurgan. *Data mining: a knowledge discovery approach.* Springer, 2007.

[14] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. The maude 2.0 system. In *International Conference on Rewriting Techniques and Applications*, volume 2706 of *LNCS*, pages 76–87. Springer, 2003.

[15] Flavio Corradini, Alessio Ferrari, Fabrizio Fornari, Stefania Gnesi, Andrea Polini, Barbara Re, and Giorgio O. Spagnolo. A Guidelines Framework for Understandable BPMN Models. *Data & Knowledge Engineering*, 2017.

[16] Flavio Corradini, Fabrizio Fornari, Andrea Polini, Barbara Re, and Francesco Tiezzi. A formal approach to modeling and verification of business process collaborations. *Sci. Comput. Program.*, 166:35–70, 2018.

[17] Flavio Corradini, Chiara Muzi, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. Animating Multiple Instances in BPMN Collaborations: From Formal Semantics to Tool Support. In *Business Process Management*, volume 11080 of *LNCS*, pages 83–101. Springer, 2018.

[18] Flavio Corradini, Chiara Muzi, Barbara Re, Lorenzo Rossi, and Francesco Tiezzi. Global vs. local semantics of BPMN 2.0 or-join. In *SOFSEM*, volume 10706 of *LNCS*, pages 321–336. Springer, 2018.

[19] Flavio Corradini, Chiara Muzi, Barbara Re, and Francesco Tiezzi. A classification of BPMN collaborations based on safeness and soundness notions. *CoRR*, abs/1809.06178, 2018.

[20] Flavio Corradini, Chiara Muzi, Barbara Re, Francesco Tiezzi, and Lorenzo Rossi. MIDA: multiple instances and data animator. In *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018*, volume 2196 of *CEUR Workshop Proceedings*, pages 86–90. CEUR-WS.org, 2018.

[21] Flavio Corradini, Andrea Polini, Barbara Re, and Francesco Tiezzi. An Operational Semantics of BPMN Collaboration. In *FACS*, volume 9539 of *LNCS*, pages 161 – 180. Springer, 2015.

[22] Ana De Medeiros and Christian Günther. *Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms*, pages 177–190. DAIMI. University of Aarhus, 2005.

[23] Gero Decker, Remco Dijkman, Marlon Dumas, and Luciano García-Bañuelos. Transforming BPMN diagrams into YAWL nets. In *Business Process Management*, volume 5240 of *LNCS*, pages 386–389. Springer, 2008.

[24] Jörg Desel. Teaching system modeling, simulation and validation. In *Winter Simulation Conference Proceedings*, volume 2, pages 1669–1675. IEEE, 2000.

[25] Remco Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281–1294, 2008.

[26] Marlon Dumas, Alexander Grosskopf, Thomas Hettel, and Moe Wynn. Semantics of standard process models with OR-joins. In *On the Move to Meaningful Internet Systems*, volume 4803 of *LNCS*, pages 41–58. Springer, 2007.

[27] Nissreen El-Saber. *CMMI-CM compliance checking of formal BPMN models using Maude.* PhD thesis, Department of Computer Science, University of Leicester, 2015.

[28] María Teresa Gómez-López, José Miguel Pérez-Álvarez, Angel Jesús Varela-Vaca, and Rafael Gasca. Guiding the creation of choreographed processes with multiple instances based on data models. In *Business Process Management Workshops*, volume 281 of *LNBIP*, pages 239–251, 2016.

[29] Christian Günther and Anne Rozinat. DISCO: Discover your processes. In *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018*, volume 940 of *CEUR Workshop Proceedings*, pages 40–44. CEUR-WS.org, 2012.

[30] Hanwen Guo, Ross Brown, and Rune Rasmussen. Virtual worlds as a model-view approach to the communication of business processes models. In *Proceedings of the CAiSE'12 Forum*, volume 855 of *CEUR Workshop Proceedings*, pages 66–73. CEUR-WS.org, 2012.

[31] Hanwen Guo, Ross Brown, and Rune Rasmussen. Human resource behaviour simulation in business processes. In *Information Systems Development*, pages 167–178. Springer, 2013.

[32] Michael Hahn, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. Modeling and execution of data-aware choreographies: an overview. *Computer Science-Research and Development*, 33:1–12, 2017.

[33] Andreas Hermann, Hendrik Scholta, Sebastian Bräuer, and Jörg Becker. Collaborative business process management - a literature-based analysis of methods for supporting model understandability. In *Wirtschaftsinformatik*, 2017.

[34] Richard Hull, Jianwen Su, and Roman Vaculin. Data management perspectives on business process management: Tutorial overview. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, page 943–948. Association for Computing Machinery, 2013.

[35] IBM. IBM Innov8 2.0. `http://www-01.ibm.com/software/solutions/soa/innov8/full.html`.

[36] IEEE. Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams - IEEE Standard, 2016.

[37] Kurt Jensen. A brief introduction to coloured petri nets. In Ed Brinksma, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 203–208. Springer, 1997.

[38] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3):213–254, 2007.

[39] Valeriia Kataeva and Anna Kalenkova. Applying graph grammars for the generation of process models and their logs. 8, 2014.

[40] Robert M. Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[41] Ahmed Kheldoun, Kamel Barkaoui, and Malika Ioualalen. Formal verification of complex business processes based on high-level Petri nets. *Information Sciences*, 385:39–54, 2017.

[42] David Knuplesch, Rüdiger Pryss, and Manfred Reichert. Data-aware interaction in distributed and collaborative workflows: Modeling, semantics, correctness. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 223–232. IEEE, 2012.

[43] Marcello La Rosa, Hajo Reijers, Wil van der Aalst, Remco Dijkman, Jan Mendling, Marlon Dumas, and Luciano GarcíA-BañUelos. APROMORE: An Advanced PROcess MOdel REpository. *Expert Systems with Applications*, 38(6):7029–7040, 2011.

[44] Ann Lindsay, Denise Downs, and Ken Lunn. Business processes attempts to find a definition. *Information and Software Technology*, 45(15):1015–1019, December 2003.

[45] Felix Mannhardt. Multi-perspective process mining. In *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018*, volume 2196 of *CEUR Workshop Proceedings*, pages 41–45. CEUR-WS.org, 2018.

[46] Jan Mendling, Hajo A. Reijers, and Wil van der Aalst. Seven process modeling guidelines (7pmg). *Information and Software Technology*, 52(2):127–136, 2010.

[47] Andreas Meyer, Luise Pufahl, Dirk Fahland, and Mathias Weske. Modeling and Enacting Complex Data Dependencies in Business Processes. In *Business Process Management*, volume 8094 of *LNCS*, pages 171–186. Springer, 2013.

[48] Andreas Meyer and Mathias Weske. Activity-centric and artifact-centric process model roundtrip. In *Business Process Management Workshops*, volume 171 of *LNBIP*, pages 167–181. Springer, 2014.

[49] Andreas Meyer et al. Data perspective in process choreographies: modeling and execution. In *Techn. Ber. BPM Center Report BPM-13-29*, 2013.

[50] Alexey Mitsyuk, Ivan Shugurov, Anna Kalenkova, and Wil van der Aalst. Generating event logs for high-level process models. *Simulation Modelling Practice and Theory*, 74:1–16, May 2017.

[51] Rocco De Nicola. A gentle introduction to process algebras, 2013.

[52] OASIS. Web Services Business Process Execution Language Version 2.0, April 2007.

[53] OMG. Business Process Model and Notation (BPMN V 2.0), 2011.

[54] OMG. Decision Model and Notation (DMN V. 1.1), 2016.

[55] Philipp Fromme and Sebastian Warnke and Patrick Dehn. bpmn-js Token Simulation, 2017. `https://github.com/bpmn-io/bpmn-js-token-simulation`.

[56] Thomas Prinz and Wolfram Amme. A Complete and the Most Liberal Semantics for Converging OR Gateways in Sound Processes. *Complex Systems Informatics and Modeling Quarterly*, 1(4):32–49, 2015.

[57] Rosario Pugliese and Francesco Tiezzi. A calculus for orchestration of web services. *Journal of Applied Logic*, 10(1):2–31, March 2012.

[58] Manfred Reichert and Barbara Weber. *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer Science & Business Media, 2012.

[59] Hajo Reijers, Irene Vanderfeesten, Marijn Plomp, Pieter van Gorp, Dirk Fahland, Wim van der Crommert, and H. Daniel Diaz Garcia. Evaluating data-centric process approaches: Does the human factor factor in? *Software & Systems Modeling*, 16(3):649–662, 2017.

[60] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press, 2008.

[61] Signavio GmbH. Signavio, 2018. `http://www.signavio.com/`.

[62] Sebastian Stamm. Creating a 3D renderer for BPMN, 2018. `https://blog.camunda.com/post/2018/02/creating-a-3d-renderer`.

[63] Thomas Stocker and Rafael Accorsi. SecSy: Security-aware synthesis of process event logs. In *Workshop on enterprise modelling and information systems architectures*, pages 71–84. Citeseer, 2013.

[64] Anna Suchenia, Tomasz Potempa, Antoni Ligęza, Krystian Jobczyk, and Krzysztof Kluza. Selected Approaches Towards Taxonomy of Business Process Anomalies. In *Advances in Business ICT: New Ideas from Ongoing Research*, pages 65–85. Springer, 2017.

[65] Bernhard Thalheim, Ove Sorensen, and Egon Börger. On Defining the Behavior of OR-joins in Business Process Models. *Journal of Universal Computer Science*, 15(1):3 – 32, 2009.

[66] OMG UML. Unified modeling language. *Object Management Group*, page 105, 2001.

[67] Wil van der Aalst. Process mining. *Communications of the ACM*, 55(8):76–83, 2012.

[68] Wil van der Aalst, Arya Adriansyah, Ana De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, et al. Process mining manifesto. In *International Conference on Business Process Management*, pages 169–194. Springer, 2011.

[69] Wil van der Aalst and Arthur ter Hofstede. YAWL: yet another workflow language. *Information systems*, 30(4):245–275, 2005.

[70] Wil van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[71] Boudewijn van Dongen, Ana De Medeiros, and Lijie Wen. Process mining: Overview and outlook of petri net discovery algorithms. In *Transactions on Petri Nets and Other Models of Concurrency II*, LNCS, pages 225–242. Springer, 2009.

[72] Boudewijn van Dongen and Wil van der Aalst. A Meta Model for Process Mining Data. *EMOI-INTEROP*, 160:30, 2005.

[73] Pieter van Gorp and Remco Dijkman. A visual token-based formalization of BPMN 2.0 based on in-place transformations. *Information and Software Technology*, 55(2):365–394, 2013.

[74] Kees van Hee and Zheng Liu. Generating benchmarks by random stepwise refinement of petri nets. In *ACSD/Petri Nets Workshops*, 2010.

[75] Visual Paradigm. Business Process Design with Powerful BPMN Software. `https://www.visual-paradigm.com/features/bpmn-diagram-and-tools/`.

[76] Hagen Völzer. A new semantics for the inclusive converging gateway in safe processes. In *Business Process Management*, volume 6336 of *LNCS*, pages 294–309. Springer, 2010.

[77] Hagen Völzer. Faster Or-Join Enactment for BPMN 2.0. In *Business Process Model and Notation*, volume 95 of *LNCS*, pages 31–43. Springer, 2011.

[78] Colin Ware and Glenn Franck. Viewing a Graph in a Virtual Reality Display is Three Times as Good as 2D Diagram. In *IEEE Symposium on Visual Languages*, pages 182–183, 1994.

[79] Mathias Weske. *Business Process Management.* Springer, 2012.

[80] Petia Wohed, Wil van der Aalst, Marlon Dumas, Arthur ter Hofstede, and Nick Russell. Pattern-based analysis of UML activity diagrams. *Beta, Research School for Operations Management and Logistics, Eindhoven*, 2004.

[81] Peter Wong and Jeremy Gibbons. A Process Semantics for BPMN. In *Formal Methods and Software Engineering*, volume 5256 of *LNCS*, pages 355–374. Springer, 2008.

[82] Moe Wynn, Eric Poppe, Jingxin Xu, Arthur ter Hofstede, Ross Brown, Azzurra Pini, and Wil van der Aalst. ProcessProfiler3D: A visualisation framework for log-based process performance comparison. *Decision Support Systems*, 100:93–108, 2017.

[83] Moe Wynn, Eric Verbeek, Wil van der Aalst, Arthur ter Hofstede, and David Edmond. Business process verification-finally a reality! *Business Process Management Journal*, 15(1):74–92, 2009.

[84] Hanna Yang, Minjeong Park, Minsu Cho, Minseok Song, and Seongjoo Kim. A system architecture for manufacturing process analysis based on big data and process mining techniques. In *2014 IEEE International Conference on Big Data*, pages 1024–1029, 2014.

ACKNOWLEDGMENTS

A long time ago in Camerino ... I started a tortuous journey full of obstacles but personally rewarding. Everything was supposed to end three years ago getting the degree. I was almost ready to leave Camerino when, right at the end, someone suggested me to raise the bar even further and try to do the PhD. *Do or do not, there is no try.* The PhD was certainly not my plan for a variety of reasons which today seem frankly very stupid. Me doing the PhD? It just doesn't exist, it's not for me, and sincerely it seems a *trap*. At that moment, I was so scared because I knew that I would be dealing with the biggest obstacle: myself. I was frightened of having to attend international conferences and talk about my research in front of respected professors.

Anyway, the PhD has not only a *dark side*, and step by step I started getting more and more involved in doing research. I loved the opportunity to build my projects, to work with colleagues who have taught me a lot, and to listen to very inspiring talks made by relevant professors. I met a lot of people and I had the opportunity to visit many places. The PhD allowed me to learn a lot of new things, the most important to me is the scientific method by which now I can see the world from a more pragmatic perspective.

Now that I have reached the finish line, I end up with this thesis. It is the *proof of work* of this hard but wonderful journey. Thus, I have to thank a lot of people, starting with the professors who helped me in different ways.

First of all, I want to thank Prof. Barbara Re and Prof. Francesco Tiezzi for believing in me from the very beginning, they are great supervisors as well as endless sources of *hope*.

A very special thank you goes also to Prof. Flavio Corradini for the lot of effort he spent in keeping our university *alive*, even after a terrifying earthquake.

Another special thank you goes to Prof. Andrea Burattin, who hosted me during the visiting period at DTU University. He inspired me a lot and made me feel at home also in a place where the native call *pasta alla carbonara* a strange sauce with mushrooms and cream.

Clearly, I want to mention the members of the committee, Prof. Maurice ter Beek and Prof. Pascal Poizat, the suggestions and the comments they gave me reviewing this thesis have been very useful for finalizing it and for planning future activities.

As results become better if one works with colleagues, I want to give a huge thank you to all the members of the ProsLab team. A special mention goes to Fabrizio, Chiara, and Andrea who were also my best office mates.

Obviously, I want to say thank you to my parents and my sister that always stand by me, even if I hardly spread my doubts and my problems with them.

Last but first, since she was also mentioned in the dedication, I thank the love of my life Chi for keeping our relationship still *synallagmatic* as it was in that little room overlooking the mountains.