

Università degli Studi di Camerino

SCUOLA DI SCIENZE E TECNOLOGIE

Corso di Laurea in Computer Science (Classe L-18)



**Multilayer consistency checking in BPMN
collaboration models based on message
exchange representation**

Laureando
Lorenzo Rossi
Matricola 094417

Relatore
Prof.ssa Barbara Re

Correlatore
Prof. Francesco Tiezzi

Abstract

The increased use of OMG BPMN 2.0, Business Process Modelling Notation, as a standard in modelling business process of an enterprise and the absence of a precise semantic for this language, has led to the emergence of several modelling issues impacting on the quality of the resulting business processes.

Here the focus is on the modelling of sub-process element in collaboration model, which is widely used to represents how organizations interact each other. In particular, I aim to better understand how messages exchange protocol can be represented in collaboration model including collapsed sub-processes that are than represented via detailed model. In this regards, the OMG standard does not provide precise specification. As a consequence, modelling approaches behave differently and in most of the case the consistency between the collapsed sub-process and its detailed representation is left to the modeler's discretion.

To solve such an issue, I aim to define a novel encoding of messages exchange protocol that allows to check consistency between BPMN collaboration model layers. The proposed approach relies on using a textual notation representing message protocol. Novel reduction rules able to avoid non communicative activities within the representation are also introduced. Then is provided a procedure for checking consistency based on the minimal dual of a messages protocol. Finally, the efforts are validated using a *Java* tool, fully integrated in the *Eclipse* modeling environment, implementing the approach.

Summing up, the proposed contribution aims to provide: (i) a textual representation of message exchange in a collaboration model; (ii) a communication protocol encoding and (iii) a procedure for consistency checking considering messages exchange modelled in different layers.

Contents

1	Introduction	1
1.1	Motivations	1
1.2	Problem Statement	2
1.3	Thesis Statement and Research Questions	3
1.4	Methodology	3
1.5	Chapters Overview	3
2	Background	5
2.1	Business Process Management	5
2.2	Business Process Lifecycle	5
2.3	BPMN 2.0	7
3	Literature Review	13
3.1	Review Methodology	13
3.2	BPMN Formalization	15
3.3	Models Consistency	16
3.4	Process Similarity	17
3.5	Formal Verification	17
4	Modelling Sub-Processes in BPMN	19
4.1	Sub-Processes Modeling Approaches	19
4.2	Expanded Sub-Process Approach	20
4.2.1	Expanded Sub-Process and Collapsed Pool	20
4.2.2	Expanded Sub-Process and Expanded Pool	21
4.3	Collapsed Sub-Process Approach	21
4.3.1	Collapsed Sub-Process and Collapsed Pool	23
4.3.2	Collapsed Sub-Process and Expanded Pool	23
4.4	Multilayer Collapsed Sub-Process Approach	24
4.4.1	Multilayer Collapsed Sub-Process and Collapsed Pool	25
4.4.2	Multilayer Collapsed Sub-Process and Expanded Pool	25
4.5	Approaches Comparison	28
5	Consistency Checking via Messages Protocol	31
5.1	BPMN Restrictions	31
5.2	Textual Representation of Messages Protocol	32

5.2.1	BPMN Encoding	32
5.2.2	Reduction Methodology	34
5.3	Multilayer Consistency Checking	37
5.3.1	Dual Model Derivation	37
5.3.2	Consistency Checking	39
6	Implementation of Consistency Checking Method	45
6.1	Architectural Overview	45
6.2	BPMN Model Parsing	47
6.3	Consistency Checking	50
7	Validation	55
7.1	Order Fulfillment Validation Scenario	55
7.2	Consistency Checking via Tool Application	57
	Conclusions and Further Development	60
	Bibliography	68

List of Figures

2.1	BPM lifecycle [10]	6
2.2	Considered BPMN 2.0 elements	8
2.3	Collaboration model: abstract level [10]	10
2.4	Collaboration model: detailed level [10]	11
4.1	Expanded sub-process approach	20
4.2	Expanded sub-process and collapsed pool modeling approach	21
4.3	Expanded sub-process and expanded pool modeling approach	22
4.4	Collapsed sub-process approach	22
4.5	Collapsed sub-process and collapsed pool modeling approach	23
4.6	Collapsed sub-process and expanded pool modeling approach	24
4.7	Sub-process model approach	25
4.8	Collapsed sub-process and collapsed pool abstract layer	26
4.9	Collapsed sub-process and collapsed pool detailed layer	26
4.10	Collapsed sub-process and expanded pool abstract layer	27
4.11	Collapsed sub-process and expanded pool detailed layer	27
5.1	(a) Unstructured model, and (b) its equivalent well-structured version [35]	32
5.2	Encoding example	34
5.3	(a) Input pattern and (b) reduction output	36
5.4	(a) Input pattern and (b) reduction output	36
5.5	Resulting model after reduction	36
5.6	The dual model	38
5.7	M_1 Model	40
5.8	M_2 Model	40
5.9	Reduced M_1 model	42
5.10	Reduced M_2 model	42
5.11	$\dot{\mathbb{E}}_1$ and $\dot{\mathbb{E}}_2$ equivalence	43
6.1	Class diagram	46
6.2	BPMN model example (a) and its model graph (b)	47
6.3	Encoding array example	47
7.1	Case Study: abstract layer with collapsed sub-processes and expanded pools	56
7.2	Case Study: <i>Acquire raw materials</i> detailed layer	57

7.3	Console output	59
7.4	\mathbf{ARM}_{Dual} model	61
7.5	\mathbf{SUPP}_{Dual} model	61

Listings

6.1	Xml to Encoding translation	48
6.2	getDual method	50
6.3	reduce method	50
6.4	Consistency checking method	53
7.1	Running example main method	58

1. Introduction

Here in the following is introduced the problem that I am dealing with. First, the reasons behind this research both from an academic and an industrial point of view are explained. Then, the problem statement and the research questions from which this thesis is derived are introduced. Lastly, the structure of the thesis is provided.

1.1 Motivations

Business process modeling is very important in understanding processes and in sharing this knowledge of the processes with the people who are involved. Indeed, process participants typically perform specialized activities in a process such that they are hardly confronted with the complexity of the whole process. Therefore, process modeling helps to better understand the process and to identify and prevent issues.

Business process models are represented basically with notational languages. There exists several notational languages that represents processes such as BPEL (Business Process Execution Language) [27] and UML (Unified Modelling Language) [31]. However, here the focus is on BPMN (Business Process Modeling Notation) [30]. BPMN is a fairly complex language with over 100 elements representing actions, synchronizations, choices and so on. One of this elements is the sub-process which represents a secondary process nested in the overall model. Moreover BPMN supports three main categories of processes: *Orchestration*, *Choreography* and *Collaboration*. Orchestration models imply a single coordinating point of view. While an orchestration process describes a process within a single business entity. Lastly, a choreography process model is a definition of the behavior between interacting participants.

The focus is on collaboration models due to their capability to describe communications among the participants. More precisely, I concentrate on sub-processes due to their practical importance in modeling that generally is a fundamental phase of the business process management lifecycle. The use of modelling guidelines make the modelling more easy and the resulting model more understandable [22]. In this the use of sub-process plays a relevant role. In fact, “*the use of sub-processes in large process models is an important step in modeling practice to handle complexity. While there are several advantages attributed to such a modular design, including ease of reuse, scalability, and enhanced understanding*” [37]. Such as the modeler should try to keep models as small as possible. Large process models are difficult to read and comprehend. Additionally, they tend to contain more errors. Defining the correct scope of tasks and level of detail of processes is a key point to reduce the overage of information [23].

To improve its readability, BPMN give this possibility by hiding certain parts within a sub-process. The sub-process can be in a collapsed view that hides its details or a sub-process can be in an expanded view that shows its details within the view of the

model in which it is contained.

For this reasons, the designer should create a hierarchical business process model with multiple levels of details for the process [39]. BPMN sub-processes can be used to split the process into “layers”. The modeler can introduce at higher layer collapsed sub-process that can be than expanded later to expose details of lower levels of hierarchy. The behavior of the sub-process is, in this sense, represented at abstract level in the main layer and detailed layer in the lower layer.

Introducing communication in this layered structure it is possible using message flows. This is particularly relevant with respect to collaboration. Moreover, the designer can introduce communications in each layer of details. In particular, are considered those modeling approach where message exchange among participants has been implemented at sub-process level. This may cause the absence of consistency between the messages exchange of related layers. Consistency means that messages exchange between participants can be performed: each message sent by a participant can be received by the other one, and vice-versa. In fact, each communication participant exchange messages, through message flows, following its messages protocol.

The method of showing messages exchange could works at different layers according to the modeling approach followed to model sub-processes: (a) on the abstract layer when the sub-process is expanded, (b) on the abstract layer when the sub-process is collapsed and (c) on the detailed layer describing sub-process diagram.

The option (a), in which the sub-process is expanded in the abstract layer, according to the mentioned guidelines is usually not recommended, since the use of collapsed sub-process make the model more readable. In terms of understandability, the adoption of such solution does not introduce much issues. It is easy to see which task is sending or receiving the message and also it is possible to consider if the message will always sent or not. The scenario does not change if the behaviour of the participants involved in the collaboration is well known or not.

Option (b), in which the sub-process is collapsed in the abstract layer, can be a solution in term of modeling, when it is not really important to know the detail of the sub-process itself. This scenario create several issues in order to understand the flow of messages. From the process point it is not possible to know what is the communication behaviour. The scenario change if the behaviour of the participants involved in the collaboration is well known or not.

The most fine solution is given by the option (c) where the detail of the sub-process is provided in the detailed layer. To give a complete scenario it need to refer to the abstract layer and this can be done combining option (b) and (c). The “correct” way, of showing messages sent from tasks inside a sub-process, is to include on the sub-process model the participant to which you are sending or from which you are receiving the message, which should be named identically to the same participant on the main diagram. Also the messages exchange should be consistent among layers of the model.

1.2 Problem Statement

As I already mentioned in applying option (b) combined with (c) as a modeling guidelines the issues rise when it is needed to model task sending messages. In particular, if in BPMN model there is a task or more inside a sub-process that sends or receives a message with a participant, need to be properly modelled both at abstract (in the collaboration) and detailed layer (in the sub-process model).

The BPMN specification [29], do not handles this issue. In particular, OMG talks about message flow in terms of uses: “a message flow is used to show the flow of messages between two participants that are prepared to send and receive them” (paragraph 9.3 [29]). OMG considers the consistency only among different models types, collaboration, choreography and conversation, without taking into account references between sub-process in different layers.

Moreover, this problem appears also in the business process modeling tools. In fact, they suffer the absence of a message protocol specification. Consequently, they face up the problem in different way, giving their interpretation.

1.3 Thesis Statement and Research Questions

The thesis statement resulting from the problem statement is following reported.

It is possible to guarantee the multilayer consistency for the same collaboration model when it includes sub-processes exchanging messages.

Whereas the research questions derived from the thesis statement are following reported.

- Is it possible to have a explicit representation of message flows in a collaboration model, to give information of message exchange at different levels?
- It is possible to derive communication protocol from the collaboration model?
- Is it possible to check consistency among messages flows modelled in different layers?

1.4 Methodology

In this section is described the method that I have applied in order to develop the proposed thesis. In particular, our effort aims to answer to the research questions provided before. Several approach with different levels of formalization can be taken to archives the goal.

Due to this motivations, the research methodology used in this thesis follows a pragmatic approach. Hence, each step of the research is done following the final goal. The formalization of the partial results and a formal definition of the work correctness are, at this step omitted. This approach ensures to reach quicker the goal both with a base for future researches.

A formal approach may requires a big effort to define and prove results supporting each step of the research. Moreover a formal approach may not converge to an answer for the research questions. In this regard, the aim is to formalize the result provided in this thesis as future work.

1.5 Chapters Overview

The chapters of the thesis are organized as follows. chapter 2 provides some background information and specification on BPMN 2.0. chapter 3 regards the literature review

showing the current state-of-the-art. Following, in chapter 4, are proposed several sub-process modeling approaches in order to analyze the information available in the models. Chapter 5 presents our consistency checking method, introducing the necessary ingredients for achieve the goal. In chapter 6 is shown the tool implementing the proposed method, while in chapter 7 the tool is used in order to validate the proposed method in a case study.

2. Background

This chapter presents some background information about business process. This should help the reader giving basic understanding of the research problem being investigated and promotes confidence in the overall quality of the following analysis and findings.

In particular, the first section provides a general introduction on business process management (BPM). The second section describes briefly the business process lifecycle. While in the third section the business process management notation is introduced in details.

2.1 Business Process Management

A business process is described as “*a collection of related and structured activities undertaken by one or more organisations in order to pursue some particular goal. Within an organisation 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 organisations*” [20].

Business processes derive from the observation that each product or service provided by a company is the outcome of a number of activities performed. Business processes are the key instrument to organizing these activities and to improving the understanding of their interrelationships.

Moreover 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 functionality that realizes new products or services.

Business process management covers, from one side, the representation of business processes, but implies also additional activities. Hence “*business process management includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes*” [20].

2.2 Business Process Lifecycle

BPM is characterized by a set of activities that occurs cyclically in order to adapt and improve the model. Hence, BPM can be viewed as continuous cycle, Figure 2.1,

comprising the following phases: identification, discovery, analysis, implementation, monitoring and controlling.

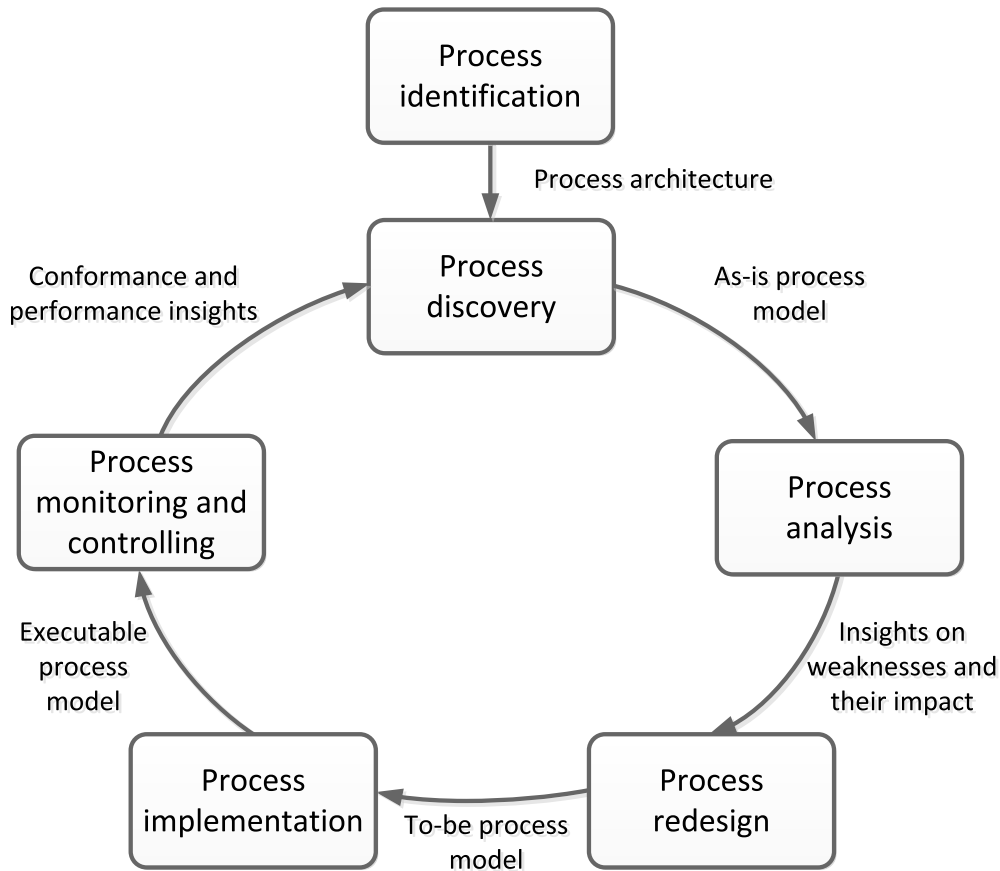


Figure 2.1: BPM lifecycle [10]

- **Process identification.** In this phase, a business problem is identified. The outcome of process identification is a new or updated process architecture that provides an overall view of the processes in an organization and their relationships.
- **Process discovery.** 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.** In this phase, issues associated to the as-is process are identified. The output of this phase is a collection of issues.
- **Process redesign.** The goal of this phase 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.
- **Process implementation.** In this phase, the changes required to move from the as-is process to the to-be process are prepared and performed.

- **Process monitoring and controlling.** Once the redesigned process is running, relevant data are collected and analyzed to determine how well is the process performing with respect to its performance measures and performance objectives.

Business process models are important at various stages in several phases of the BPM lifecycle. Before starting to model a process, it is very important to understand why we are modeling it. There are many reasons for modeling a process. The main one regard the understandability of the process and the possibility to share our understanding of the process with the participants who are involved with the process.

Indeed, process participants typically perform specialized activities in a process such that they are hardly confronted with the complexity of the overall process. Therefore, process modeling helps to better understand business processes and to identify and prevent issues.

2.3 BPMN 2.0

Many different languages and graphical notations have been proposed to represent business process models with differences both in the possibility to express aspects related to the perspectives, and in the level of formality used to define the elements composing the notation. BPMN 2.0¹, which has been standardised by OMG [28], is currently acquiring a clear predominance, among the various proposal, due to its intuitive graphical notation, the wide acceptance by industry and academia, and the support provided by a wide spectrum of modelling tools².

BPMN's success comes from its versatility and capability to represent business processes with different levels of detail and for different purposes. The notation acquired, at first, acceptance within business analysts and operators, who use it to design business process models. Successively, it has been more and more adopted by IT specialists to lead the development and settlement of IT systems supporting the execution of a BP model. Among the various characteristics of the notation, particularly interesting is the possibility to model a *collaboration* of different organisations exchanging messages and cooperating to reach a shared business goal.

Collaboration models are indeed the focus of our work since they contains enough information to describe participants behavior, and the message flow specified to permit successful co-operations. Collaboration models are essentially a collection of pools that represent the participants. In a BPMN process model there is usually only one participant. In BPMN collaboration models, you can have multiple processes or participants represented on the model. You can also model the messages that are sent between the processes and how they interact with each other. The exchange of messages between different pools is represented by message flows. In this model, pools can be collapsed or expanded, in the latter case the pool include processes.

Following, I do not aim to provide a complete presentation of the standard, but a discussion of the main concepts of BPMN collaboration model used in the following. These concepts are briefly described below and reported in Figure 2.2.

- **Pools** are used to represent participants or organizations involved in the collaboration, and provide details on internal process specifications and related elements. Pools are drawn as rectangles.

¹BPMN or BPMN 2.0 are used interchangeably to refer to version 2.0 of the notation.

²BPMN is currently supported by 75 tools (see <http://www.bpmn.org> for a detailed list).

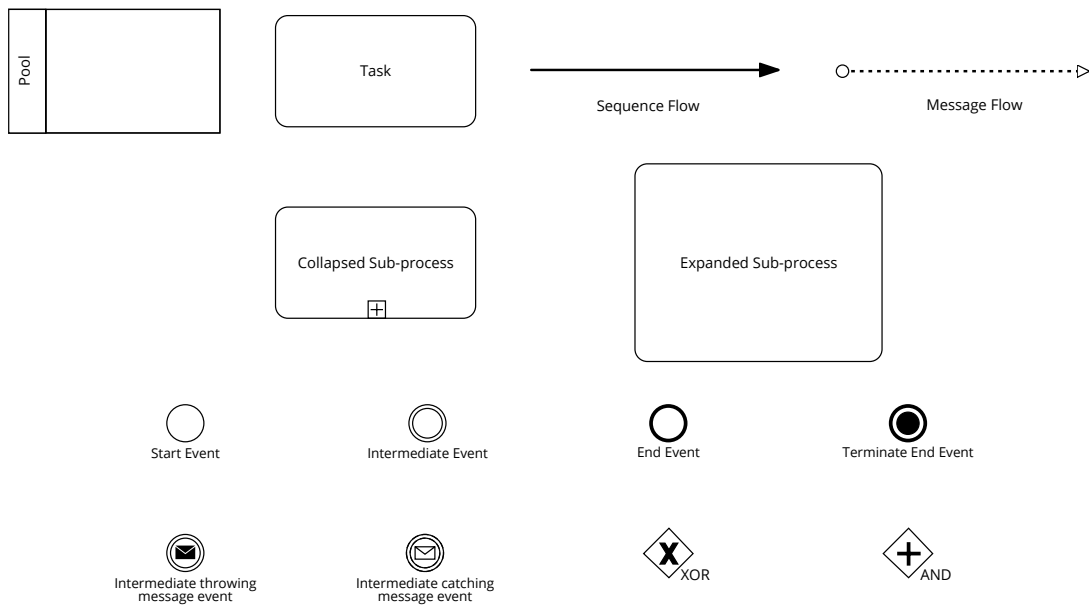


Figure 2.2: Considered BPMN 2.0 elements

- **Tasks** are used to represent specific works to perform within a process. Tasks are drawn as rectangles with rounded corners.
- **Connecting Edges** are used to connect process elements in the same or different pools. *Sequence Flow* is used to specify the internal flow of the process, thus ordering elements in the same pool, while *Message Flow* is a dashed connector used to visualise communication flows between organisations.
- **Sub-Process** is an activity that represents, in a higher detail level, another process. It is represented, graphically speaking, as a task within a process. The internal details of a Sub-process is modeled using Tasks, Gateways, Events, and Sequence Flows, in order to show it's behaviour. Sub-processes are useful to increase the understandability of a model with many activities, they can describe the detailed behaviour of a process that is not very important at the main process level. There are two main kind of Sub-Process, collapsed or expanded, Figure 2.3.
 - A collapsed sub-Process object has the same shape as the Task object: a rounded rectangle with a marker to distinguish it as a Sub-Process, rather than a Task. The marker is a small square, positioned at the bottom center of the shape, with a plus sign + inside. In this case there is not a detail of the behaviour of the sub-process.
 - An expanded sub-Process object is a rounded rectangle. The behaviour of the sub-process is provided inside the rectangle and it is modelled as a collaboration model.

Moreover sub-processes could be divided in other sub categories: ad-hoc sub-process, looped sub-process or event sub-process. An ad-hoc sub-process is a specialized type of sub-process that is a group of activities that have no sequence relationships. An event sub-process is a specialized sub-process that is triggered by events. A looped sub-process is a specialized type of sub-process that is performed more than one time.

- **Events** are used to represent something that can happen. An event can be a *Start Event* representing the point in which the process starts, an *Intermediate Event* representing something happened during process execution, while an *End Event* is raised when the process terminates. Events are drawn as circles. When an event is source or target of a message flow, it is called *Message Event*. According to the different kinds of message flow connections, it is possible to observe the following situations.
 - *Start Message Event* is a start event with an incoming message flow; the event element catches a message and starts a process.
 - *Throw Intermediate Event* is an intermediate event with an outgoing message flow; the event element sends a message.
 - *Catch Intermediate Event* is an intermediate event with an incoming message flow; the event element receives a message.
 - *End Message Event* is an end event with an outgoing message flow; the event element sends a message and ends the process.

I also refer to a particular type of end event, the *Terminate End Event*, displayed by a thick circle with a darkened circle inside. It is able to stop and abort the running process.

- **Gateways** 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 flows) or split nodes (forking into outgoing sequence flows). Different types of gateways are available.
 - A *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 “×”.
 - An *AND gateway* enables parallel execution flows. An AND-split gateway is used to model the parallel execution of two or more branches, as all outgoing sequence flows are activated simultaneously. An AND-join gateway synchronises the execution of two or more parallel branches, as it waits for all incoming sequence flows to complete before triggering the outgoing flow. An AND gateway is drawn with a diamond marked with the symbol “+”.

A multilayer collaboration example is provided following in order to show a real case of an order fulfillment process that is divided into two layers. The first abstract layer in Figure 2.3 provides the pools for the seller and the a supplier. In this layer are present two sub-processes: one collapsed describing the raw material acquisition and one expanded regarding the shipment of the invoice. The collapsed sub-process element behaviour, that resumes the raw materials acquisition, is not provided in this layer, however the model shows that there exist a messages exchange with the supplier. Differently, *Ship and invoice* activity is represented in a detailed way showing the model in it. The detailed layer, Figure 2.4, gives a more clear representation of the action performed to prepare the order. In fact, the seller firstly check the availability of stock

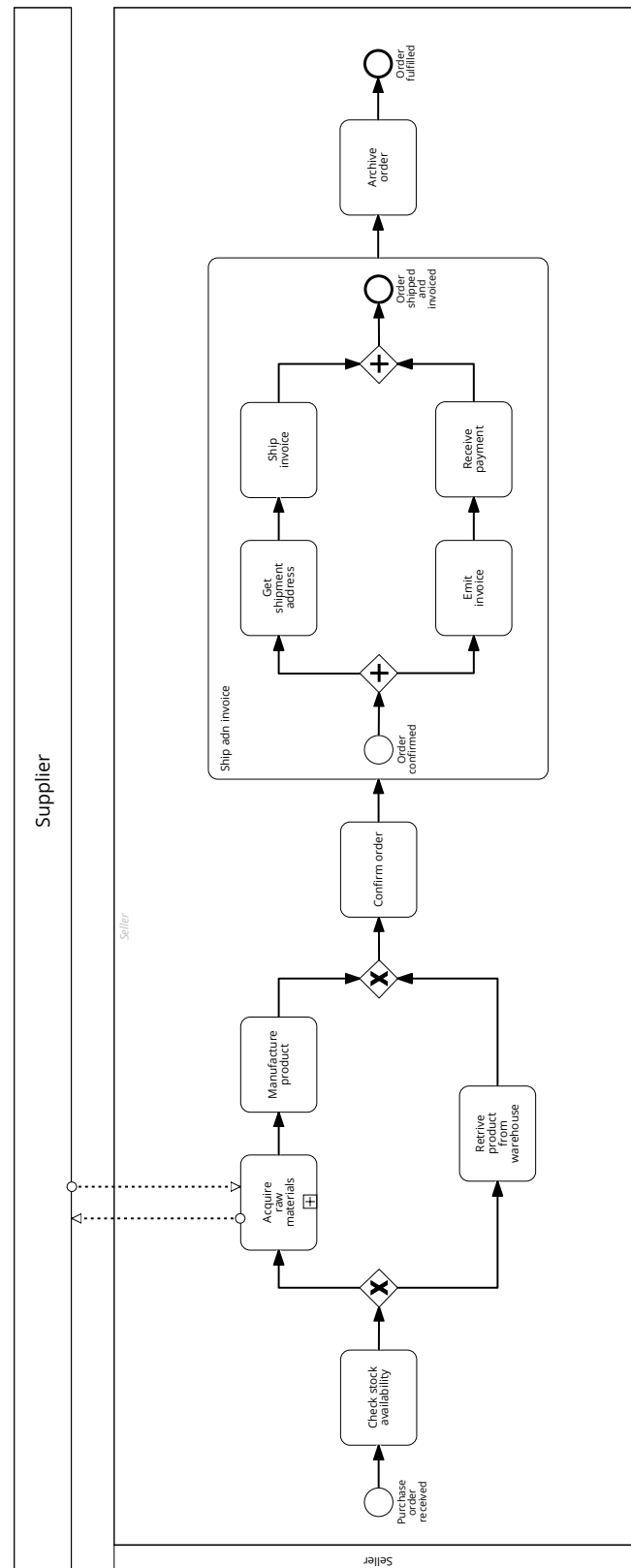


Figure 2.3: Collaboration model: abstract level [10]

material, then, if this material is not available, is requested to supplier. More in detail, there are two communicative tasks, one sending to the supplier the material request, an one receiving back the ordered material.

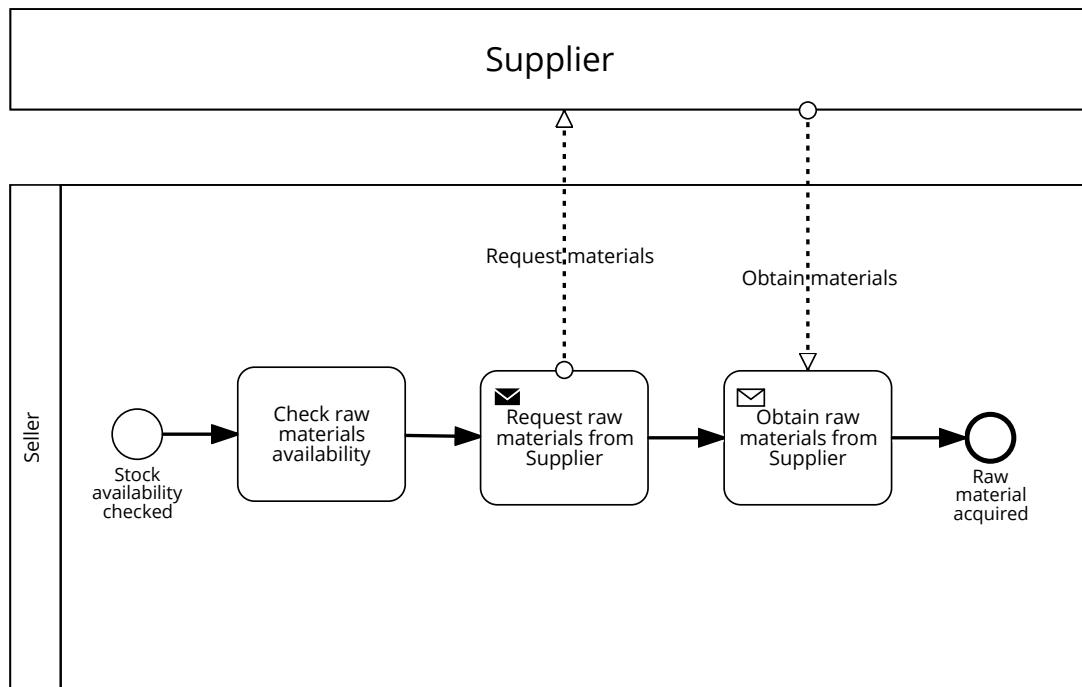


Figure 2.4: Collaboration model: detailed level [10]

3. Literature Review

This chapter presents a review on the existing literature concerning multilayer consistency in business process modelling. The goal of this chapter is to provide an overview of existing literature and the current state-of-the-art. This information is needed, on one hand, to reinforce the thesis statement and, on the other hand, to develop new concepts by combining techniques and methods from different areas. Relevant areas of the literature review are introduced briefly.

In particular, the first section explain the methodology used to collect the research papers used in this thesis. The second section presents the academic effort in formalizing a semantics for BPMN with regards to messages exchange in sub-processes, while in the third section provides an overview on multilayer consistency in a less strict point of view. Lastly, in the fourth section, the focus is on the techniques for detecting similar processes with regard to simplification methods, while the fifth section analyzes the efforts provided in formal verification works.

3.1 Review Methodology

The literature review was done on according to the snowballing method as described in [43]. In particular, in order to identify the set of relevant research papers to consider in the next phases, the method suggests to start from an initial set of papers manually identified according to some criteria. Successively further relevant papers can be identified proceeding backward and forward in time, using respectively the related works section of already included papers (backward snowballing), and the “cited by” functionality provided by digital libraries¹ (forward snowballing).

In this case, a set of initial papers is identified running a manual search on conference proceedings and journals that according to our personal knowledge have an high reputation within communities for which the definition of consistency checking for collaboration models can be certainly considered a relevant topic.

Therefore the scientific program for the following international conferences were analyzed:

- Advanced Information Systems Engineering;
- Business Information Systems;
- Business Process Management;
- On the Move to Meaningful Internet Systems;

While considered journals are:

¹<https://scholar.google.it/>

- Data & Knowledge Engineering;
- Decision Support Systems;
- Information and Software Technology;
- Information Systems;
- Information Systems Frontiers.
- Business Information Systems Engineering
- Enterprise Information Systems
- Information System Modeling and Design
- International Journal of Cooperative Information Systems
- International Journal of Simulation and Process Modelling
- Journal of Systems and Software

A advantage of manual search respect to a search on digital libraries is the possibility to retrieve recent publications that are generally not already indexed by library.

An additional criterion considered to shape the initial set of papers refers to the publication date. In particular, I have initially considered only papers published after the release of the BPMN 2.0 specification, i.e. January 2011.

In order to include or not a paper published in the mentioned venues I have initially considered the title and the abstract, then, when promising, I have proceeded reading at least the introduction and the conclusions sections.

As a result the initial set was constituted by several papers that were then thoughtfully read. A second step of the method asks to run backward and forward snowballing. This step was not limited neither to the selected conferences and journals nor to the considered time frame. This allow to have a comprehensive picture on related research. The reference list of the identified research papers is checked looking for possible relevant studies not considered so far. In order to decide if a paper was relevant or not I have proceeded again as described for the initial set, i.e. with successive reading of title, abstract, introduction and conclusions. As a result other further relevant works to be included in the relevant set are identified.

Backward and forward snowballing search was done in an iterative way also on the paper identified in the second step. I stopped after two iterations since I did not find additional relevant research papers. Details on selected papers included in the method application final result will be presented following. Than, I run a categorization activity in order to make easier the successive steps of collection synthesis and homogenization. Papers were then grouped in the four following categories, Table 3.1.

Formalization	[5] [7] [9] [11] [18] [42]
Consistency	[6] [8] [12] [17] [19] [26] [33] [38] [41]
Similar Processes	[4] [34] [40] [46]
Formal Verification	[1] [2] [3] [14] [45]

- **Formalization.** It refers to papers presenting sub-processes messages exchange formalization.

- **Consistency.** It refers to papers presenting consistency checking in multilayer architectures.
- **Similar Processes.** It refers to papers presenting similar processes analysis techniques.
- **Formal Verification.** It refers to papers presenting formal verification techniques of business processes.

3.2 BPMN Formalization

In order to face up the problem of multilayer consistency, a survey on BPMN formalizations is needed. In fact, the work on formalising the BPMN standard, faced ambiguities and inconsistencies, regarding both syntactical and semantic definitions in the BPMN standard.

For this reason, different interpretation of the standard comes out. In fact, OMG do not provides a rigorous semantics for BPMN elements, the meaning is given with descriptions in natural language, so that BPMN can be used in different contexts. Unfortunately, semantic definitions of the BPMN standard are in several cases inaccurate, incomplete or inconsistent.

Hence, in the last years, a relevant effort has been devoted by the research community to provide a formal semantics to the BPMN notation. Here the focus is on those formalization considering sub-processes and messages exchange in order to better understand their semantics.

Unfortunately many papers regarding BPMN formalization do not consider this particular topic. In fact, Christiansen et. al. define a formal semantics for a minimal subset of BPMN elements containing just the inclusive and exclusive gateways and the start and stop events [7]. El-Saber and Boronat also restrict their effort giving a formal characterization and semantics specification of well-structured processes [11]. Lastly, Borger and Thalheim define an extensible semantic framework for business process modeling notations [5] without taking into account sub-process messages exchange.

Differently, Van Gorp and Dijkman define a formalization using visual transformation rules. The formalization consists of rules that are documented visually using BPMN syntax. In-place transformations update models directly and do not require mappings to other languages. Moreover, they used a tool to develop a implementation of all rules. The result is a promising complement to the standard, in particular because all rules have been extensively verified and because conceptual validation is facilitated [42].

However, also this papers do not take into account sub-processes in terms of messages protocol, collaboration or consistency checking. It is clear that sub-process semantics is developed without taking into account consistency in message exchange.

In fact, Kossak et. al. propose a sub-process semantics. The paper skip the problem of messages flow saying that “*semantics, however, does not change with the graphical depiction, that is, a collapsed sub-process must have the same semantics as when it is expanded*” [18]. The focus is on the execution semantics of the sub-process, in particular referring to the possible triggers of its execution.

Also Dijkman proposes a mapping from BPMN to Petri nets, for which efficient analysis techniques are available. The paper introduces also sub-processes saying that “*the behaviour of such a process is however not clear in the BPMN specification*”. It

focus on the exception handling in sub-process execution making several simplifications [9].

From this survey it is obvious that the absence of a messages protocol formalization in sub-process modeling approach avoid consistency checking.

3.3 Models Consistency

Consequently to this survey on BPMN formalization, I focus on consistency at messages exchange level. However consistency is not a specific matter of business process modeling. It is a quality feature that denotes the absence of errors within a model or a set of models. There exists several research largely focuses on checking consistency of individual model and of relationships between pairs of models [38]. So that, our investigation bring us to focus on a less specific approach that do not regards directly consistency in terms of message protocol.

A similar results is reported by Conforti et. al. [8]. In particular, this paper aims to present a technique for multilayer discovery of BPMN models without considering the issue of message protocol.

Other related research efforts are those in the area of models consistency. In particular, Padberg et. al. discuss the maintenance of the model consistency as consequence of changes. In this paper they present a layered architecture for modeling workflows in ad-hoc networks of mobile devices that communicate with each other. Workflows are modeled using a layered architecture in which rises the question of consistency [33].

An important contribution coming from the area of abstraction of business process models. Smirnov et. al. capture the same business process on different abstraction levels. It is possible by grouping activities according to the data flow of the model in a meaningful way. Unfortunately, the paper does not provide a specific approach to abstract message flows [41]. However, the paper introduce an important structural decomposition of a process model known as the *Refined Process Structure Tree* (RPST). The RPST parses a process model into a hierarchy of fragments, each having a SESE node. The containment hierarchy of these fragments forms a tree structure, the RPST.

In this sense, consistency checking require behavioural analysis techniques that may result easier considering structured process models. In fact, well-structured business process models provides recurring structures that are firstly easier to comprehend and less error-prone than unstructured ones [19]. Moreover, well-structured models are easier to layout, understand, support, and analyze [35].

With this regard, García-Bañuelos proposes an approach to systematically identifying and classifying sub-graphs in a BPMN model that may be translated to BPEL (Business Process Execution Language) code. The partition method divide the BPMN model into SESE regions and then perform a sort of reachability analysis with an iterative method to gather control flow information. Moreover the paper defines *structured blocks* as SESE regions within a split and a join gateway [12].

Köpke et. al. shows an approach for implementing projections of abstract inter-organizational business processes [17]. They first design an abstract global process, then each task of the global process is assigned to one of the partners. Finally, a partitioning procedure is applied to generate views of the global process for each partner. The partners can then develop or adopt their private processes based on their views. They proposed an algorithm for the automatic partitioning of any block-structured process with any arbitrary assignment of partners and prove its consistency.

Similarly, Norta and Eshuis defines a formal framework for specifying structurally harmonized business process collaborations. The framework permits verification of harmonized processes before their enactment. Moreover, the framework uses private and public layers to protect competitive knowledge of the individual partners [26].

Finally, in the field of software engineering, Bracciali et. al. manages the problem of adapting components that exhibit mismatching behaviour [6]. It is a crucial problem in component-based software engineering. They use process algebras to descriptions communication protocols, and it enables more sophisticated analysis of concurrent systems.

3.4 Process Similarity

Another important study field, regarding our purpose, comes from the structural similarities of processes. In order to determine these parts of a process model that are consistent, one needs to analyze collections of process models, and identify similar behaviours as consistent messages exchange protocol. In these terms the emerging research has done great steps towards the process models similarities by focusing on three different streams such as text semantics, structural analysis and behavioral analysis of the process models.

Skouradaki et. al. examine the problem of the automated detection of re-occurring structures in a collection of process models, when text semantics or behavioral data are missing [40]. This problem is a case of graph isomorphism, which is mentioned as NP-complete in the literature. Since they consider process models are very special types of attributed directed graphs, they are able to develop an approach for the detection of re-occurring structures in any process models collection, and validate it against a set of BPMN models.

From a different point of view, Pittke et. al. propose a novel approach for the detection of equivalent process model parts is proposed [34]. This approach includes three conditions of similarity based on the activity labels, flows labels and the behavioural comparison. Pittke et. al., inspired by the efforts done in [44], allow to check the similarity of two processes in terms of meaning, differently from our goal.

Wynn et. al. proposed an approach that relies on using formal methods to determine the correctness of business processes with cancellation and OR-joins [46]. The paper also demonstrates how reduction rules can be used to improve the efficiency in verification. The goal of this paper is to demonstrate that process verification has matured to a level where it can be used in practice. A useful rule, provided in the paper, is the one called *irreducible cancellation regions*. This rule individuates pieces of the model that can be not considered in determine correctness.

Baumann et. al. present a paper that provides a easily computable method for calculating behavioral similarity values for process models. The focus of the work lies on the behavioral aspect of process models, i.e., on control flow and how two models can be compared with respect to this aspect [4].

3.5 Formal Verification

The effort provided in literature in order to a semantic to BPMN could result useful for model, implement, execute and monitor business process models. With this regard,

Groefsema and Bucur provide a survey in formal business process verification [14]. This paper gathers and analyzes several works related to this topic considering their ability to check soundness, compliance and variability properties.

In this sense, Wong and Gibbons present a process semantics in the language of Communicating Sequential Processes (CSP) [15] for a subset of BPMN. Such a semantics allows developers to formally analyse and compare BPMN diagrams and assert correctness conditions that can be verified using a model checker. The proposed semantic also allows *compatibility* of collaboration between multiple business processes to be verified. The authors define *compatibility* between BP participants as “*a necessary behavioural property for a successful collaboration: the mutually consistency of the assumptions each makes about their interaction*”. This property is verified on communicating models by using a model checker, but unfortunately they do not take into account consistency in multilayer architecture containing sub-processes. However, the authors assert that “*to the best of our knowledge, no work has been done towards the compatibility verification of business collaborations described in a graphical modelling notation like BPMN*” [45].

Similarly, Anderson et. al. model e-Business processes and their properties of interest in a process algebra language, CSP [1]. This paper demonstrates how model checking can aid in the design and assurance of business processes in environments characterized also by communication. Unfortunately, this work relies on the use of UML sequence diagrams, instead BPMN.

Arbab et. al. describe how BPMN models can be represented by means of a semantically precise channel-based coordination language called *Reo* which admits formal analysis using model checking and bisimulation techniques. The paper provides also the representation, hence the verification, of message flows and sub-processes. In particular, they assert that if a process or a sub-process that must react to a message flow is not ready to accept it, the current sequence flow will be blocked [2]. In this sense, messages consistency is not verified a priori, but in case of inconsistency they present such an exception handling procedure.

Awad et. al. introduces an approach for automated compliance checking [3]. Compliance rules are translated into temporal logic formulae that serve as input to model checkers which in turn verify whether a process model satisfies the requested compliance rule. Even though, this work manages only compliance rules. However, in order to avoid the problem of state-space explosion the authors introduce an employment of a set of reduction rules. This rules allows to reduce the model size in order to guarantee a faster verification.

4. Modelling Sub-Processes in BPMN

This chapter consist of a survey on the approach that may occur in modeling BPMN collaboration. The aim of this chapter is to analyze the modeling approach, discussing about the information available in the models.

In particular, the first section introduces the possible different approaches of sub-processes design. Then, in the second, third and fourth sections are analyzed the plausible models that comes from this approaches. In the last section, the provided approaches are discussed and compared.

4.1 Sub-Processes Modeling Approaches

BPMN leaves designer free to create models with arbitrary topology, using any provided element and in a hierarchical way with several layers of detail. Hence it is necessary to identify a list of plausible modeling approaches of BPMN collaboration that include communicating sub-process elements.

Hence the focus is on collaboration models with at least two pools to allow communications and a sub-process element that exchange messages. The model can bring multiple layers in which are specified the main model, abstract layer, and the sub-process model, detailed layer.

In design collaboration models it is possible to observe several uses of sub-process element. In addition to the collapsed and expanded sub-process, there exists the possibility to use ad-hoc sub-process, looped sub-process or event sub-process. Each of those elements can communicate with one or more pools and may have an exception flow.

Moreover, where defined, the behaviour of a sub-process has no restrictions, it is possible to have more than one start or end event and the use of the gateway is up to the designer.

These modelling approaches are basically the ones introduced previously that describe sub-process:

- (a) Expanded sub-process in the abstract layer;
- (b) Collapsed sub-process in the abstract layer;
- (c) Sub-process model in the detailed layer.

This approaches regard sub-process specification in a collaboration model. However there are several other modeling approaches that can be derived. In fact, the designer

could include a detailed layer modeling sub-process and enrich the model by expanding the second pool in the abstract layer.

Consequently it is possible to identify a wider list of modeling approaches. Following, they are listed and analyzed in order to discuss about research questions of communication protocol and consistency checking. Each modelling approach following provided is discussed taking care of the consistency in sub-process communication, that is the capability to perform a communication protocol consistent with the other participant one. Moreover, is discussed the necessity to derive the communication protocol of a not specified model.

4.2 Expanded Sub-Process Approach

The first modelling approach consists in an abstract layer containing an expanded sub-process, Figure 4.1. This modeling approach provides explicitly, in the abstract layer, the behaviour of the sub-process that is fully represented both with the message flows. This approach, intuitively, do not need a detailed layer defining the sub-process behaviour. However, using an expanded sub-process element is not recommended due to improve the overall understandability of the business process model.

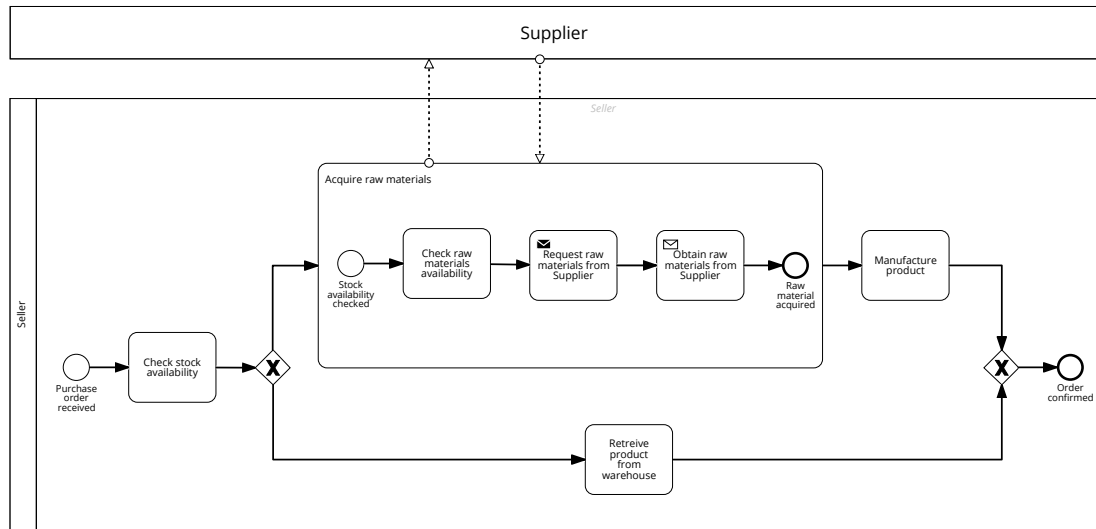


Figure 4.1: Expanded sub-process approach

The approach using expanded sub-process can be managed differently, depending on the definition of the collapsed pool in the model. Basically, this model approach do not requires a detailed layer in which describe the sub-process behaviour, due to the presence of its entire definition in the abstract layer. Hence, using expanded sub-process, two plausible approaches are characterized following.

4.2.1 Expanded Sub-Process and Collapsed Pool

This modelling approach shows a main BPMN model with one expanded pool named *A* and one collapsed pool named *B*. The main process contains a expanded sub-process element in *Pool A*, named *SUB-PROCESS*. The element *SUB-PROCESS* contains its model definition and provides communications, in both direction, with *Pool B*, each of the four message flow is provided explicitly.

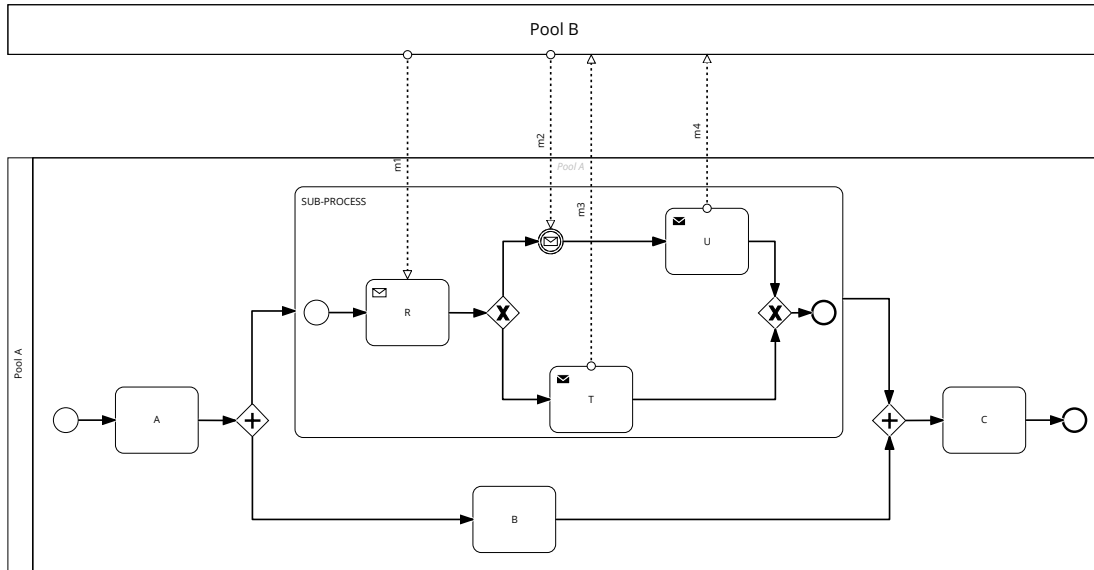


Figure 4.2: Expanded sub-process and collapsed pool modeling approach

The model of *Approach A1* presents a single layer definition in which are represented the sub-process explicitly. Hence the detailed representation of the message exchange gives all the information about the message protocol. However, due to the absence of a specification for *Pool B*, there is not the need of checking consistency.

4.2.2 Expanded Sub-Process and Expanded Pool

This approach shows a main BPMN model containing two expanded pools, named *A* and *B*. *Pool A* has a expanded sub-process element, *SUB-PROCESS*, that communicates with *Pool B*. The four message flows are provided explicitly and refers to individual elements of *SUB-PROCESS* and *Pool B*.

Here, the behaviour of messages can be derived both in *SUB-PROCESS* and *Pool B*. The example in Figure 4.3 shows intuitively that message behaviour of *SUB-PROCESS* is not consistent with the *Pool B* one. In fact, once task *L* sends message *m1*, *R* receives the same message, than *SUB-PROCESS* choose a sequence flow after the *XOR-Split*. Hence tasks *M* and *N*, that are executed in parallel in *Pool B*, can not communicate both. However checking consistency may result more difficult in models with higher dimension.

4.3 Collapsed Sub-Process Approach

Another approach consists in use collapsed sub-processes in the abstract layer, Figure 4.4. This approach also provides the behaviour of the abstract layer. Differently, the sub-process element is collapsed in this approach. Hence the representation of the sub-process model is hidden. The presence of message exchange is shown by flows between the pool and the sub-process element, but it is not fully specified due to the absence of *SUB-PROCESS* model.

The use of collapsed sub-process also entails two plausible approaches that depends on the presence of a collapsed or expanded pool describing *Pool B*. Differently from

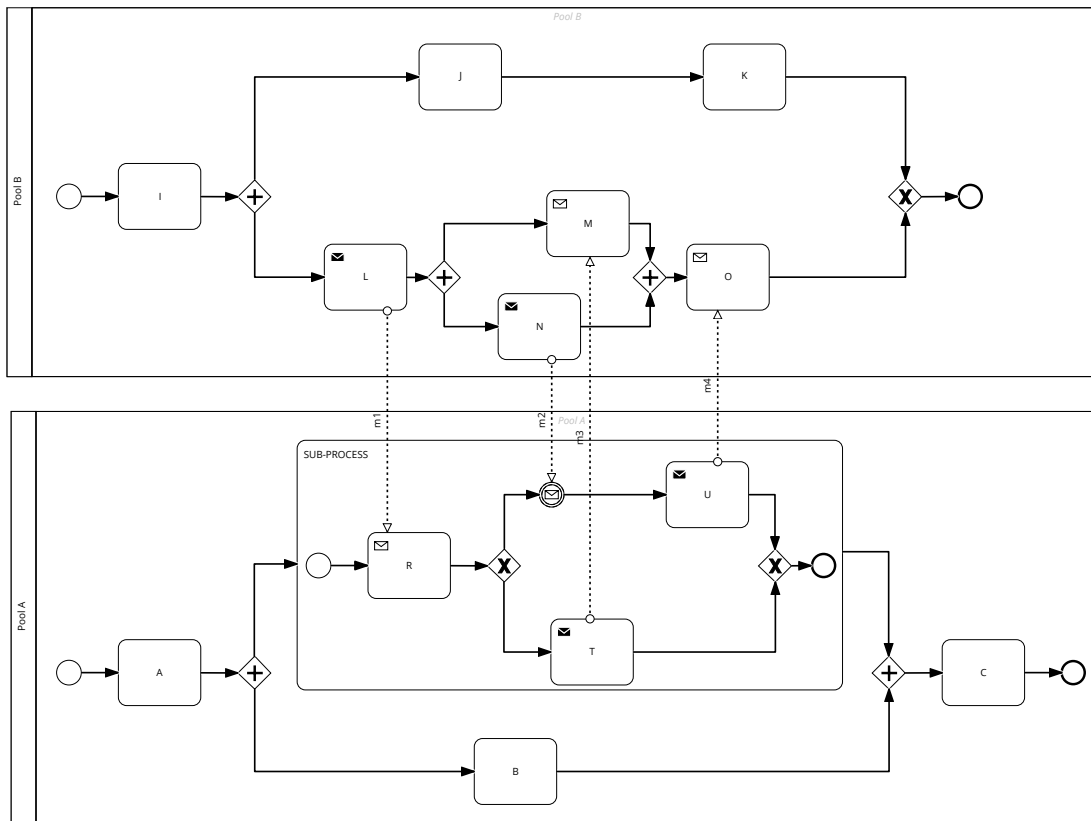


Figure 4.3: Expanded sub-process and expanded pool modeling approach

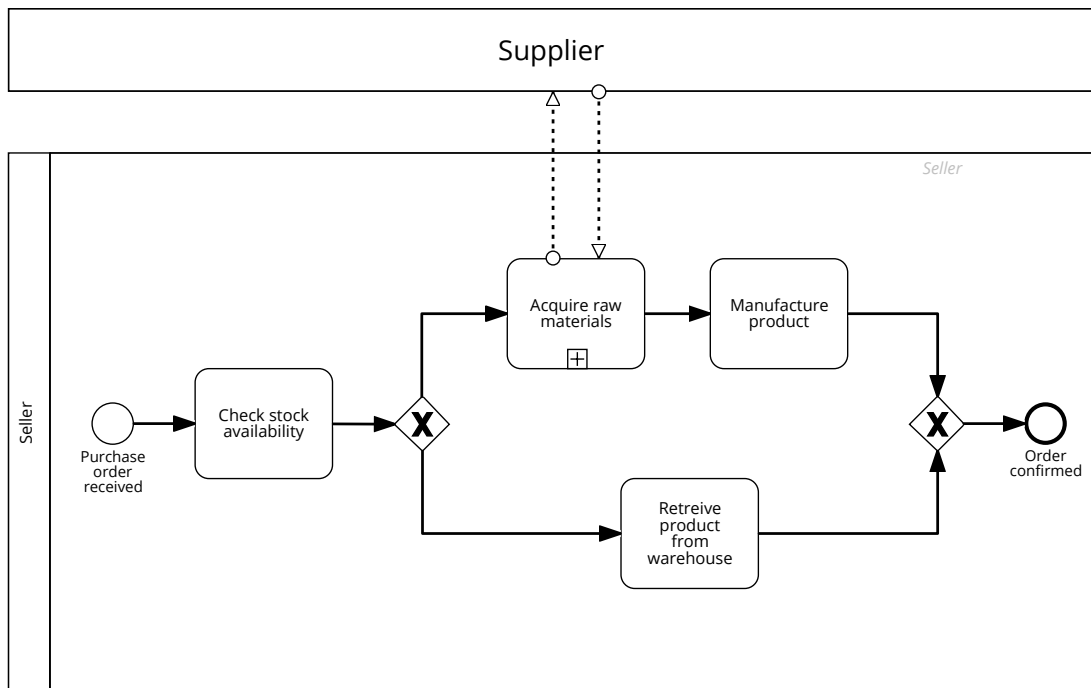


Figure 4.4: Collapsed sub-process approach

approach with expanded sub-process, the collapsed one may involve the presence of a

detailed layer representing the *SUB-PROCESS* model.

4.3.1 Collapsed Sub-Process and Collapsed Pool

This approach shows a abstract BPMN collaboration layer with one expanded pool named *A* and one collapsed pool named *B*. The abstract layer contains a collapsed Sub-Process element in *Pool A*, named *SUB-PROCESS*. The element *SUB-PROCESS* communicates, in both direction, with *Pool B*, each of the four message flow is provided explicitly.

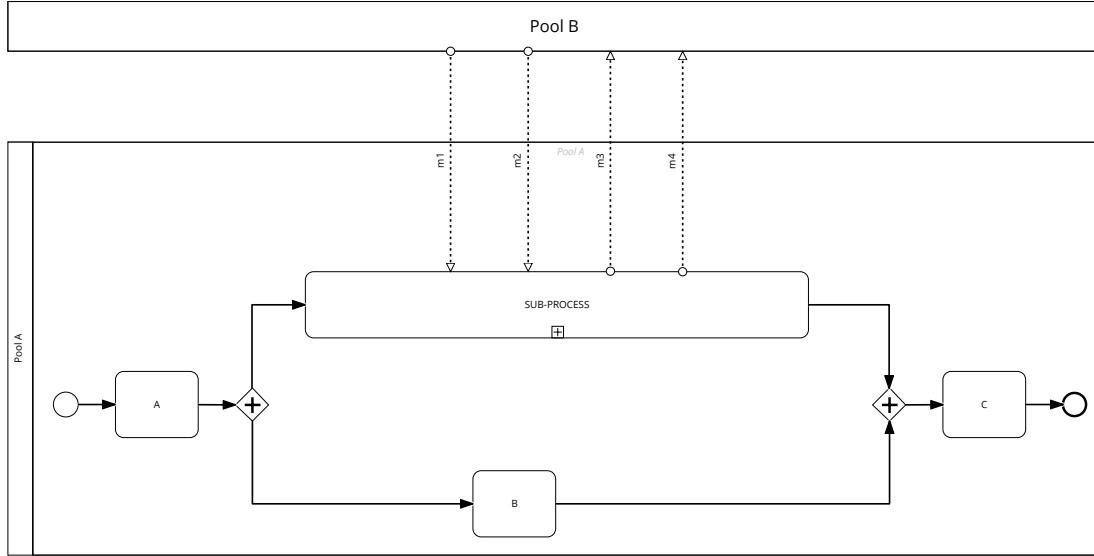


Figure 4.5: Collapsed sub-process and collapsed pool modeling approach

The model, Figure 4.5, presents an abstract layer in which are represented four message flows without any behavioural information. In fact, a priori, it is impossible to know the message protocol. So that it is interesting to derive it automatically. But the only information extractable from the model are: the communicating pools, the number of message exchanged and the direction of the message flow. There exists infinitely many models for *SUB-PROCESS* that can be consistent with the abstract layer. A lot of additional information, useful to model *SUB-PROCESS*, are missed: the BPMN elements associated to each message flow, the sequence of execution of the messages and so on. We may associate to *SUB-PROCESS* any model which presents four message flows.

4.3.2 Collapsed Sub-Process and Expanded Pool

This approach shows an abstract BPMN layer containing two expanded pools, named *A* and *B*. *Pool A* has a collapsed sub-process element, *SUB-PROCESS*, that communicates with *Pool B*. The four message flows are provided explicitly and refer to individual elements of *Pool B*.

The model, Figure 4.6 introduces more details, in particular the behaviour of *Pool B* is explicitly represented. So that the message protocol is provided, in particular is known that first there is the occurrence of message *m1*, than *m2* and *m3* follows in parallel. Lastly *m1* is received in *Pool B*. Hence it is reasonably possible to obtain

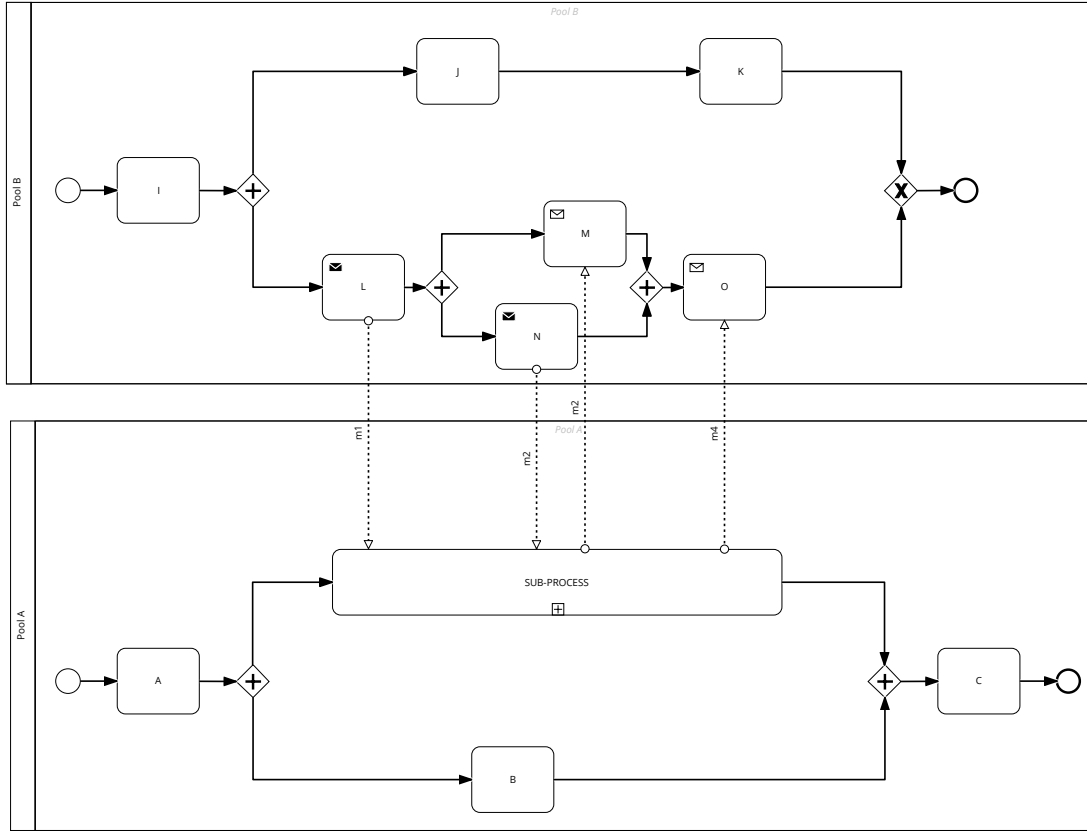


Figure 4.6: Collapsed sub-process and expanded pool modeling approach

information about the behaviour of *SUB-PROCESS*. The topology of the piece of model which communicates from *Pool B* suggests which messages *SUB-PROCESS* have to send or receive and in which order. Hence the sub-process model have to be, in such way, dual. Be dual means that *SUB-PROCESS* have to perform actions complementary to the ones performed by the tasks *L*, *M*, *N* and *O*. Hence the *SUB-PROCESS* message protocol got intuitively four elements which exchange messages: a first element receiving message *m2*, two elements in parallel, one that send *m3* and one that receive *m4*. Lastly a element sending message *m1* have to appear.

4.4 Multilayer Collapsed Sub-Process Approach

The last approach consist in a detailed layer with the sub-process specification, Figure 4.7. This approach provides a representation of a sub-process model in a detailed layer. This representation shows the complete behaviour of the sub-process both with the messages exchange. Here is missing the definition of a abstract layer in which present a collapsed sub-process element that refers to the model in the detailed layer.

The approach of using a detailed layer to represent a sub-process model, imply the presence of an abstract layer. In fact, a detailed layer have to refer to a collapsed sub-process element in the abstract layer. Hence the approaches following provided present both abstract and detailed layer.

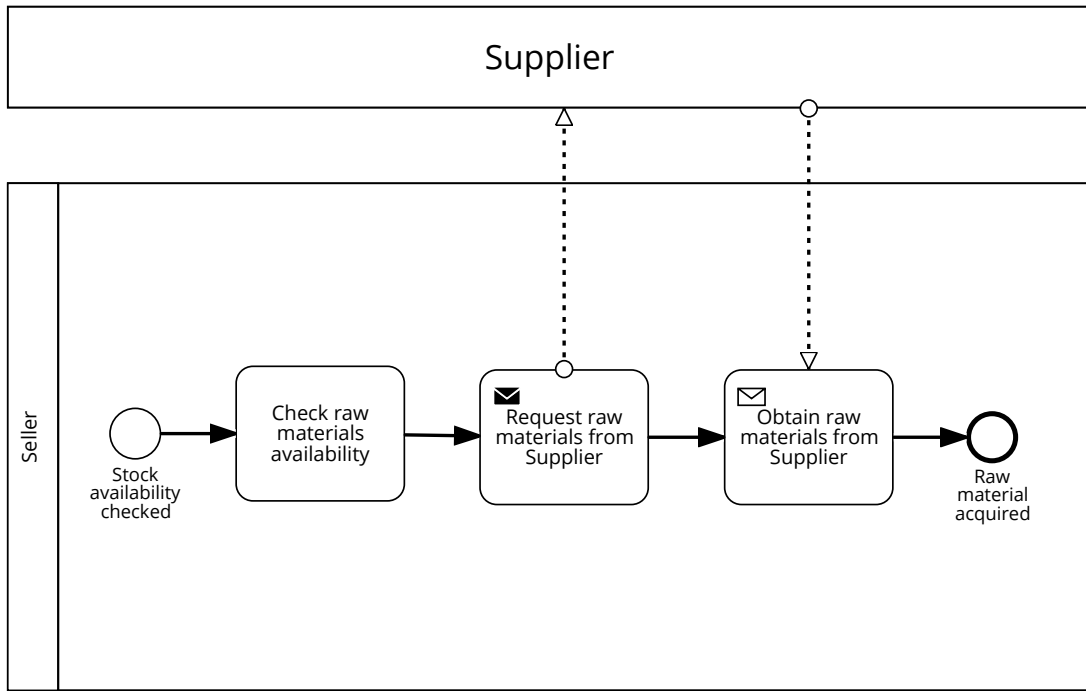


Figure 4.7: Sub-process model approach

4.4.1 Multilayer Collapsed Sub-Process and Collapsed Pool

This approach shows an abstract BPMN model with one expanded pool named *A* and one collapsed pool named *B*. The main process contains a collapsed sub-process element in *Pool A*, named *SUB-PROCESS*. The element *SUB-PROCESS* communicates, in both direction, with *Pool B*. Furthermore the detailed business process model of *SUB-PROCESS* is provided, it has the same pools, *Pool A* and *Pool B*, as the main model and shows each message flow individually sent or received by task or event.

In this approach is available a complete representation of *SUB-PROCESS*, Figure 4.9, that provides the message protocol. This approach in contrary to the previous one, provides the message flows in the detailed layer of definition. Hence the information derived from the specification of *SUB-PROCESS* allow to speculate on the behaviour of *Pool B*, Figure 4.8. On the other hand OMG does not specify how model this behaviour in the main layer. For instance the modeler may draw only two message flows with opposite directions between the collapsed sub-process and *Pool B*. Another possibility is to draw each message flow, four in our approach, but, however, information about the possible execution of those messages are undefined.

4.4.2 Multilayer Collapsed Sub-Process and Expanded Pool

This approach shows a main BPMN model containing two expanded pools, named *A* and *B*. *Pool A* has a collapsed sub-process element, *SUB-PROCESS*, that communicates with *Pool B*. The four message flows are provided explicitly and refers to individual elements of *Pool B*. Furthermore the detailed business process model of *SUB-PROCESS* is provided, it has the same pools, *Pool A* and *Pool B*, as the main model and shows each message flow individually sent or received by task or event.

This approach shows a detailed specification of each layer of the business process

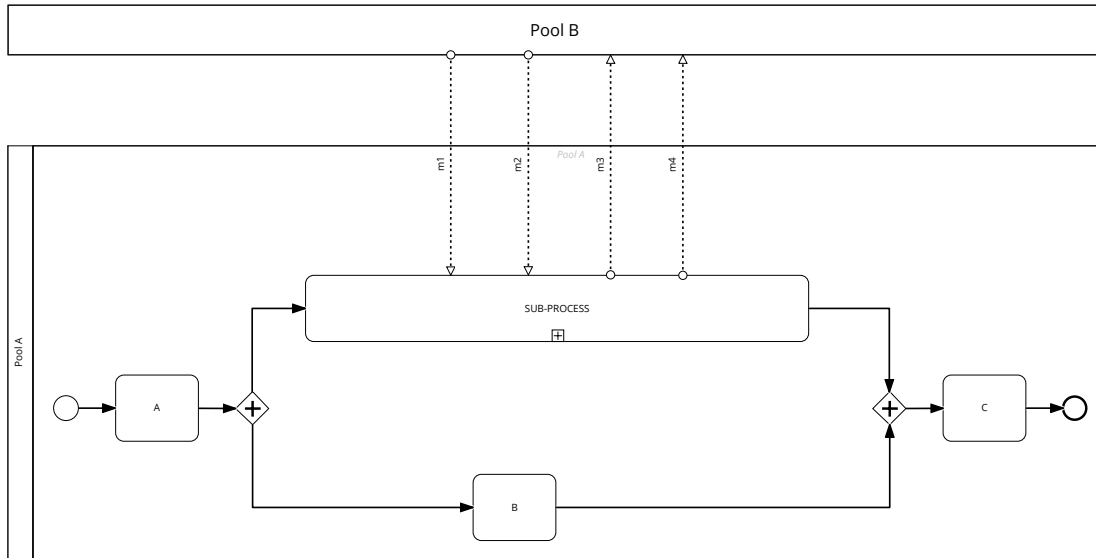


Figure 4.8: Collapsed sub-process and collapsed pool abstract layer

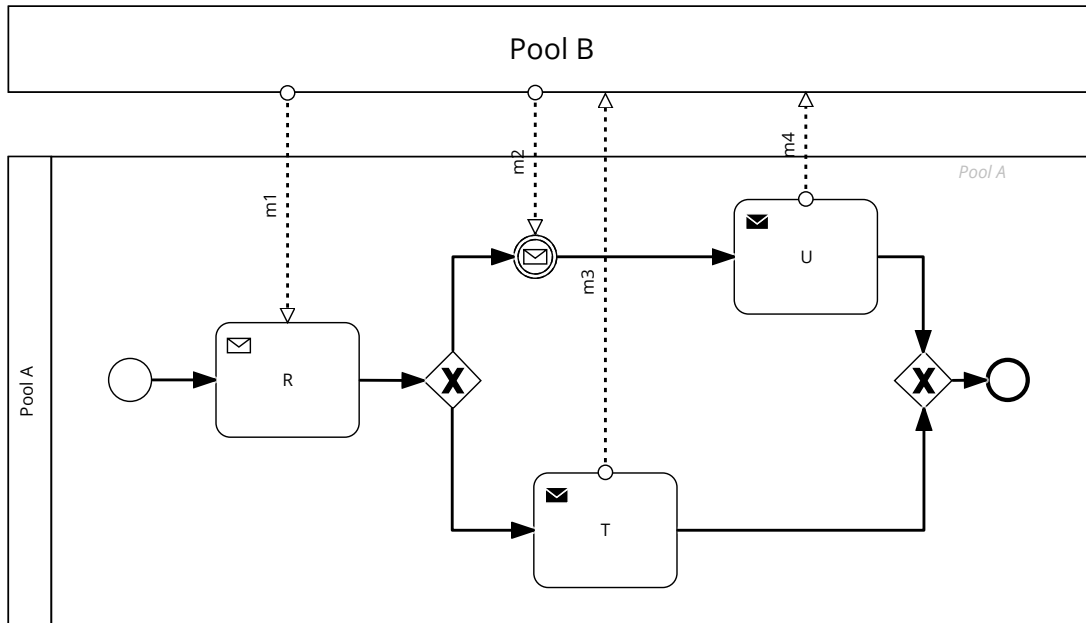


Figure 4.9: Collapsed sub-process and collapsed pool detailed layer

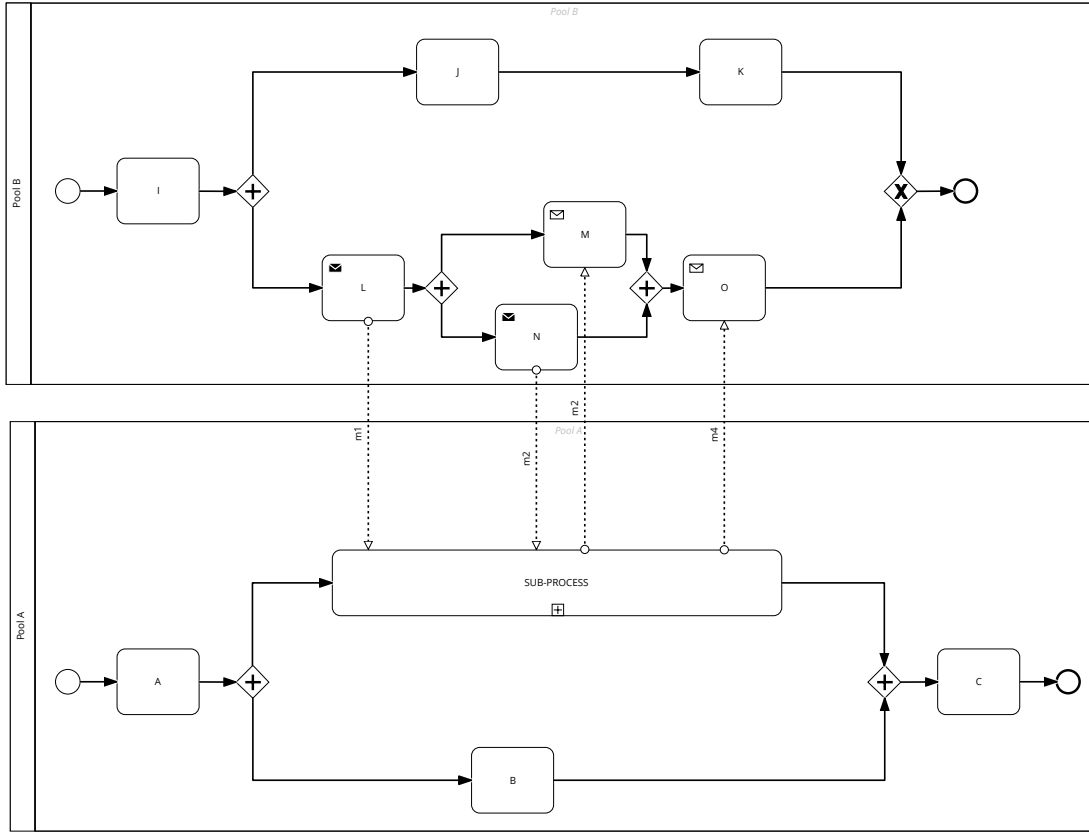


Figure 4.10: Collapsed sub-process and expanded pool abstract layer

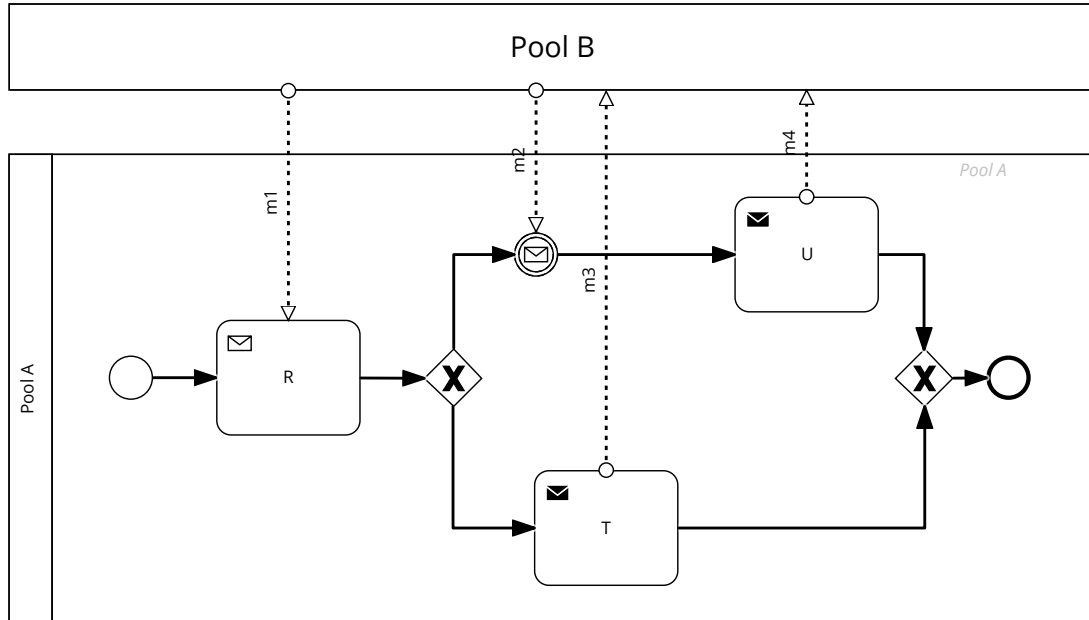


Figure 4.11: Collapsed sub-process and expanded pool detailed layer

model. Hence the message protocol can be deduced from *Pool B* and the *SUB-PROCESS* model. Here the goal is to know if the specification of *SUB-PROCESS* is consistent in the main layer. This can be done checking if every communication of *Pool B* can be replied by *SUB-PROCESS*. The example in Figures 4.10, 4.11 can be manually analyzed due to its small size: task *L* in *Pool B* sends out a message *m2* while task *R* in *SUB-PROCESS* receives a message *m2*. Then in parallel *M* and *N* send and receive *m4* and *m3* respectively but, *SUB-PROCESS* choose if *S* receives *m4* or *T* sends *m3*. Hence the two layer could not communicates properly. Consequently to this trivial analysis it is sure that *SUB-PROCESS* is not consistent in the main layer.

More in general to know whether or not a sub-layer is consistent in the main layer is a useful information in order to reuse one consistent model for describe the behaviour of *SUB-PROCESS*.

4.5 Approaches Comparison

The analysis done for this six approaches underlines the lack of information about the message protocol of a piece of model. There exist approaches in which the sub-process behaviour is provided explicitly both with the pool that communicates with it. Instead, other approaches do not provide the sub-process behaviour, however is known the behaviour of the pool that communicates with it, or vice-versa only the pool behaviour is known. And also exists approaches in which sub-process behaviour is missing and also the pool that communicates with it is not provided.

This approaches differs in the information that are acquirable from them. In fact, once a communication participant behaviour is fully provided, it is possible to know its message protocol and consequently the other participant have to fit this protocol. In this sense, approaches that provide both participants messages protocol, allows to check consistency between this communicating models. While approaches with the definition of one of those protocols do not give enough information to derive the unknown dual.

Table 4.1 shows for each approach, the derivable information and if is feasible to check consistency. Approaches are listed one per row, while each column shows if the message protocol of the participants is (\mathbb{P}) provided or (\mathbb{D}) derivable and if consistency between them is verifiable (\mathbb{V}).

			Participants		Consistency
			Sub-Process	Pool B	
Approaches	Exp. S-P	Coll. Pool	\mathbb{P}	\mathbb{D}	
		Exp. Pool	\mathbb{P}	\mathbb{P}	\mathbb{V}
	Coll. S-P	Coll. Pool			
		Exp. Pool	\mathbb{D}	\mathbb{P}	
	M Coll. S-P	Coll. Pool	\mathbb{P}	\mathbb{D}	
		Exp. Pool	\mathbb{P}	\mathbb{P}	\mathbb{V}

Table 4.1: Approaches comparison table

Intuitively, the approach using a collapsed sub-process and a collapsed pool, due the complete absence of participant messages protocol definitions, no piece of model can be derived. So that, this approach does not allow consistency checking. On the other hand the approaches that use one expanded sub-process and pool in the detailed layer and one a multilayer architecture with the definition of each participants, allow

to compare the two participants messages protocols in order to check consistency. The remaining approaches define just one participant between sub-process and tha pool that communicate with it. However the not provided messages protocols can be derived by fitting the defined dual participant.

In addition, these three approaches are interesting in terms of reuse. In fact, models or parts of those, may describe processes useful in other diagrams. In this cases, process models are overlapping, although parts could be easily reused saving costs and efforts. The approach using collapsed sub-process with expanded pool has, in this sense, the capability to reuse any model, consistent with the pool, as sub-process specification.

5. Consistency Checking via Messages Protocol

This chapter discusses about the possibility to check consistency in multilayer collaboration models using an encoding method for representing in a textual way the message behavior of a process model.

In particular, the first section provides some restriction of BPMN elements and topology useful to introduce, in the second section, the definition of the textual representation based on a proposed mapping of the BPMN elements and on the encoding of the message protocol. Then in the third section is shown how to derive from a model its consistent dual model and how to check consistency.

5.1 BPMN Restrictions

In order to approach the problem in a more understandable way, I restrict the considered BPMN elements and topologies.

A subset of possible multilayer BPMN collaboration models in which are provided at least two communicating pools is taken into account. If there exists a detailed layer that models the collapsed sub-process, it has to contain the same pools of the abstract layer model and have to deny multiple start or end event and looped behaviours.

However, BPMN allows process models to have almost any topology. The topology is a model property that, in this case, describes its structure. More in general topology describes how points are connected each other, in this case the points represent the BPMN elements while the connections are the sequence flows. For example, a fixed set of BPMN elements can be used to model process with different topologies.

Hence, it is often desirable that models abide by some structural rules [35]. In this sense, a BP model can be characterized by its structure. Basically a model can be or not well-structured.

A model is well-structured, if for every node with multiple outgoing flows, i.e. an and split, there is a corresponding node with multiple incoming flows, i.e. an and join, such that the piece of the model between the split and the join forms a Single-Entry-Single-Exit process component [36]. Otherwise, when it does not fit this quality the model is unstructured, Figure 5.1. More in detail Single-Entry Single-Exit (SESE) component is a piece of the model with two elements, the entry and the exit, and with other elements being reachable from the entry and being part of at least one path to the exit.

Enforcing restrictions may lead to a reduction of expressive power [21]. However a well-structured process is also well-behaved in the sense that it does not deadlock [16]. Due to the qualities of well-structured processes, there exist techniques transforming a model to its structured version.

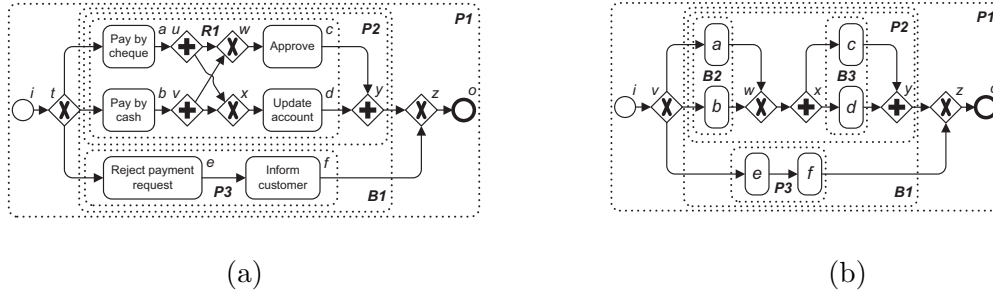


Figure 5.1: (a) Unstructured model, and (b) its equivalent well-structured version [35]

In this respect, here is taken into account also the model structure, in particular are considered only well-structured models due to its lower complexity and the presence of only SESE components in it.

5.2 Textual Representation of Messages Protocol

The approaches seen in the previous chapter underline how the lack of information about the behaviour of model parts avoid consistency checking between different layers and compromise the reuse of existing models. For this reasons the introduction of a textual representation, describing in a compact way this behaviour, is relevant.

The textual notation proposed here allow to:

- (i) derive a dual model;
- (ii) check consistency between communicative models.

Hence the goal is now to define first an understandable textual representation for the BPMN elements, then give an encoding method that allows to capture the communication protocol of a model.

5.2.1 BPMN Encoding

The main idea is to define a map in which for each considered BPMN element corresponds a unique textual representation. Moreover this translation have to keep messages protocol and discard information about tasks that do not send or receive messages. Discarding information about not communicative tasks, do not compromise the overall description of messages flow model behaviour. For instance, a model without messages exchange is consistent with any non communicative model, because neither model have to reply to a message exchange. Hence, a non communicative task do not have to provide information about the messages behaviour, and so, its description can be omitted. Consequently, this mapping identifies four kind of BPMN elements: start, end and intermediate message events and send or receive tasks. Even though the start event and the end event do not communicates at all in BPMN, their information are kept in order to maintain the textual notation understandability.

The mapping, resumed in Table 5.1, refers to the previously defined subset of BPMN elements. More in detail, the mapping function takes in input a BPMN element and returns a string. The start and end events are translated using their names, *startEvent* and *endEvent*. The tasks that do not send or receive messages are represented by τ

because their information are not relevant in our scope. Send tasks and intermediate throwing message events are mapped to the concatenation of the exclamation mark and the name of the associated message flow, $!channelName$. Vice versa the receive tasks and the intermediate catching message events have the question mark before the message flow name, $?channelName$. Lastly the translation of parallel and exclusive gateway are respectively mapped to a plus and a x between the angular brackets, $< + >$ and $< \times >$.




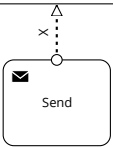
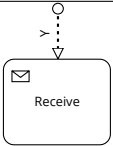
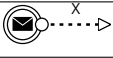
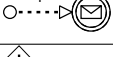


Name	Element	Mapping
Start event		$startEvent$
End event		$endEvent$
Task		τ
Send Task		$!X$
Receive Task		$?Y$
Intermediate throwing message event		$!X$
Intermediate catching message event		$?Y$
Parallel gateway		$< + >$
Exclusive gateway		$< \times >$

Table 5.1: Mapping table

Consequently to this mapping, now it is necessary to keep the model message protocol. So that, it is necessary to combine the mapped element in order to provide the behaviour of the model, for instance if two tasks run in sequence, in parallel or exclusively. This can be done encoding the BPMN model by using a intuitive representation, similar to a process algebra. Hence the behaviour of the model is represented using the encoding provided following.

Let be El , the set of model mapped elements, G , the set of mapped gateways, and $T = El \setminus G$ the set of tasks and events. Given $t \in T$, a BPMN encoding \mathbb{E} is given as follows:

As example, Figure 5.2 show a well-structured process model that behaves like follow, after the start event *Task A* is performed. Then or *Send* task and *intermediate throwing event* send a messages X and Z respectively or a message Y is received by *Receive* task both with *Task B*. After this choice a message W is received by *intermediate throwing event* and the process ends with the *endEvent*.

The resulting encoding can be derived by dividing the process in smaller *SESE* components until a single element is reached. The colored boxes in the figure are the *SESE* model components. In fact the process performs block A then B and lastly C . Intuitively each block can be recursively split, i.e. block B performs block $B1$ or $B2$ and

Name	Encoding	Description
Sequence	$t \ \mathbb{E}$	The piece of model encoded with \mathbb{E} is performed after the execution of t
And	$t < + > \mathbb{E}$	The piece of model encoded with \mathbb{E} is performed in parallel with t
Xor	$t < \times > \mathbb{E}$	The piece of model encoded with \mathbb{E} or t are performed exclusively

Table 5.2: Encoding table

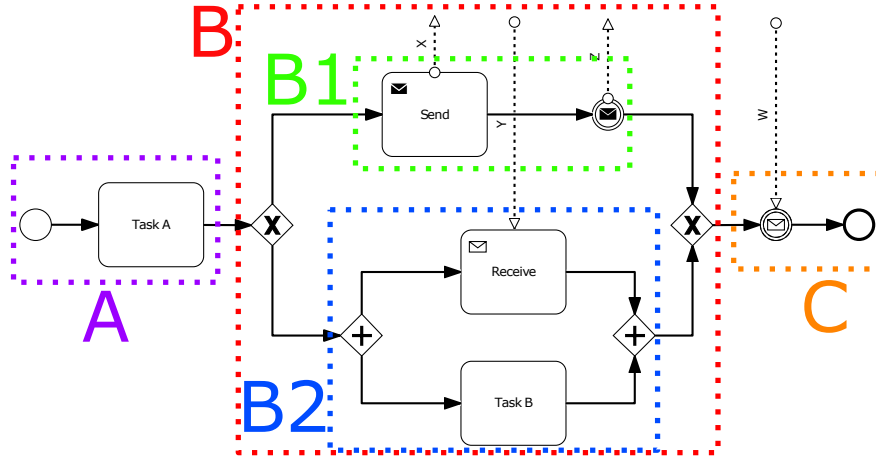


Figure 5.2: Encoding example

so on. The entire encoding is following provided in 5.1.

$$startEvent \ \tau \ (!X \ !Z < \times > (?Y < + > \tau)) \ ?W \ endEvent \quad (5.1)$$

5.2.2 Reduction Methodology

The encoding provided before gives the possibility to know intuitively the message behaviour of a model. However, encoding a BPMN collaboration model, in such way, keeps information that result unnecessary both for the textual representation and for messages behaviour.

In order to make the textual representation more compact, a method for reducing the encoding is introduced. All the task without messages exchange are encoded with the symbol τ due to their irrelevance. In this sense some of those tasks can be deleted from the representation of the message behaviour.

A list of reduction pattern can be individuated considering that I am dealing with only well-structured models and a subset of BPMN elements. Each of this reduction rule keep intact the overall messages behaviour, hence the encoding before and after the reduction have the same meaning, so they are equivalent and also the well-structure of the model is preserved. The pattern are listed following using a graphical notation¹:

More in detail, the first rule *Tau* deletes the non communicative tasks from the model. In fact, any *tau* task is removed, and consequently the incoming and the out-

¹A task named with τ represents any task without message exchange while a task named M represent a communicating element.


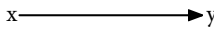
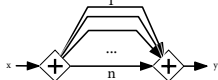
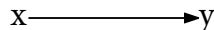
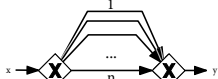
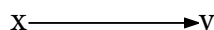
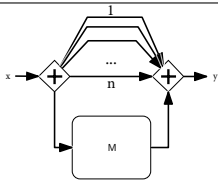

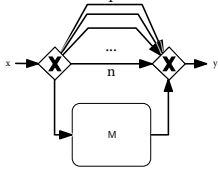
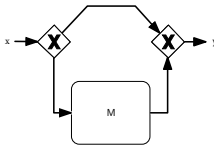
Rule	Representation	Reduction
Tau		
And_1		
Xor_2		
And_2		
Xor_2		

Table 5.3: Rules table

going sequence flow are merged. Rules *And_1* and *Xor_1* delete parallel and exclusive compositions that have one or more sequence flows connecting the split with the join gateway. The result is a sequence flow connecting the incoming flow of split element with the outgoing flow of the join one.

Rule *And_2* regards the parallel compositions with at least one communicative element within a branch. In this case all the path without elements are discarded. If there exists a unique branch that performs communications, then also the parallel composition is deleted connecting the incoming and the outgoing flows to the communicative element. Otherwise, the composition keeps all the communicative branches.

Finally *Xor_2* rule can be applied in exclusive compositions with at least one communicative element within a branch. Differently from the previous rule, in this case an empty path between split and join element, if exists, is preserved due to keep the possibility for the model to not perform communications.

The listed rules allow to reduce the model in input, in order to obtain a compact textual representation of the message behaviour. The encoding derived from the reduction avoids τ elements that are not necessary in the global meaning of the model.

The reduction procedure starts applying rule *Tau* for each non communicative element in the model. The resulting model contains only communicative elements, hence the other rules can be applied in order to avoid all those paths that do not contains elements.

So that, model in Figure 5.2 shows a first τ task, *Task A*, that can be deleted due to rule *Tau*. The reduction deletes the task and links the incoming sequence flow with the outgoing one. Figure 5.3 provides the pieces of model before (a) and after (b) the reduction.

The model presents another τ task, *Task B*, which also follows the rule *Tau*. The reduction, Figure 5.4 deletes the task as in the previous case, moreover the resulting piece of model, Figure 5.4 (b), belongs to the rule *And_2*. Hence the *and* split is deleted

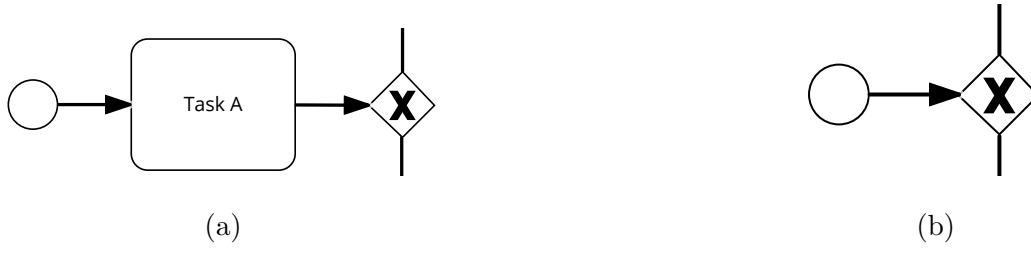


Figure 5.3: (a) Input pattern and (b) reduction output

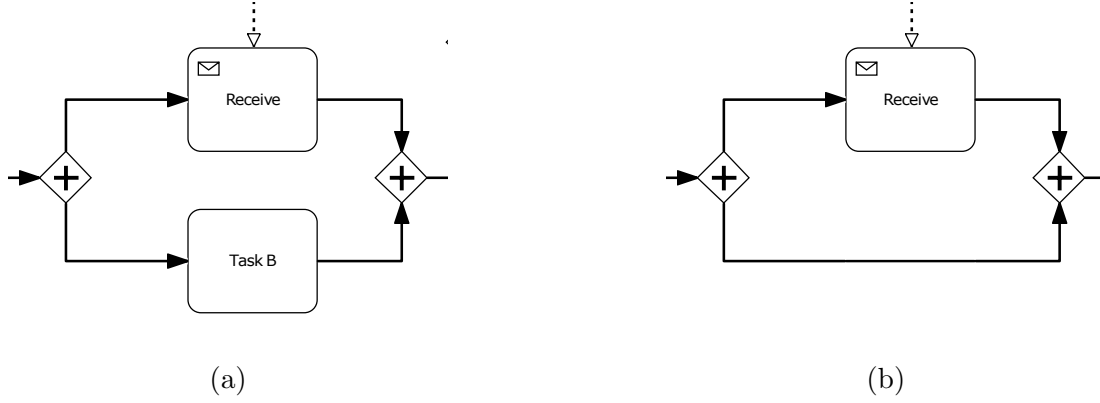


Figure 5.4: (a) Input pattern and (b) reduction output

and the model keeps only task M , producing the model in Figure 5.5.

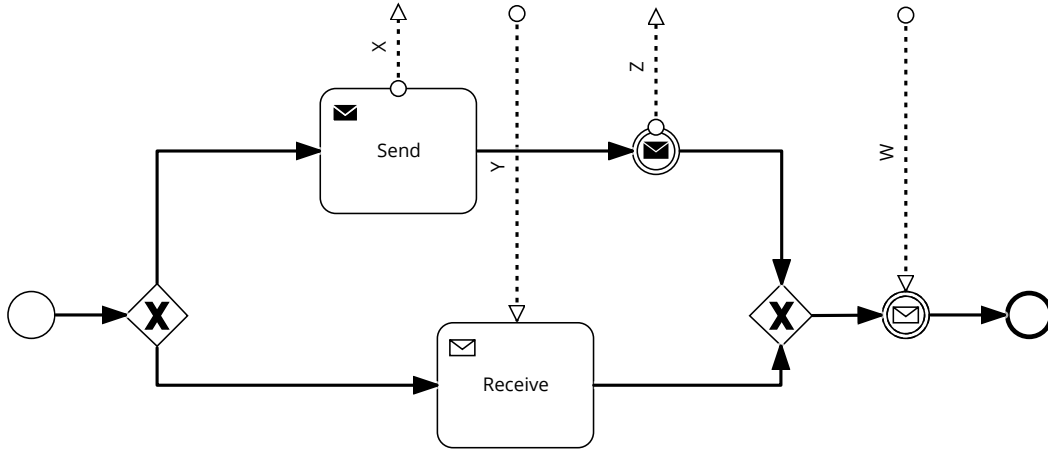


Figure 5.5: Resulting model after reduction

The textual representation of the model changes consequently the application of these rules. The diagram is free of τ elements, so that the encoding results more compact 5.2, and also it is equivalent to 5.1.

$$startEvent (!X !Z < x > ?Y) ?W endEvent \quad (5.2)$$

5.3 Multilayer Consistency Checking

The provided encoding brings, in a compact way, the message protocol of a BPMN model. Moreover, the textual representation, can be used to derive a dual model and check consistency. Here are presented in detail how the proposed encoding suggests both the behaviour of its model and the behaviour of their dual models.

Lastly consistency checking is discussed in our scenario. In detail, is provided a method for checking if two model layers are consistent.

5.3.1 Dual Model Derivation

I have shown how capture a model message protocol and how it can be represented in a understandable textual way with the provided encoding. Intuitively, a model with message exchange can not communicate with any other model, but it need a specific model that is able to fit each communication, sent or received. More precisely, whenever the model sends a message, the dual one have to receives the same message flow, and vice-versa. Hence is possible to define the dual for each element mapped into the encoding, Table 5.4.





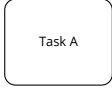
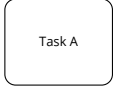
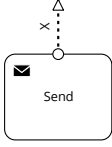
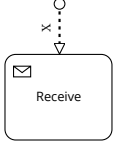
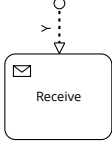
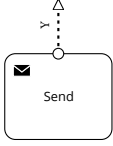
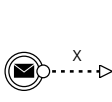
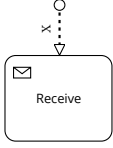
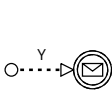
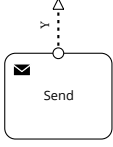




Name	Element	Mapping	Dual Element	Dual Mapping
Start event		<i>startEvent</i>		<i>startEvent</i>
End event		<i>endEvent</i>		<i>endEvent</i>
Task		τ		τ
Send Task		<i>!X</i>		<i>?X</i>
Receive Task		<i>?Y</i>		<i>!Y</i>
Intermediate throwing event		<i>!X</i>		<i>?X</i>
Intermediate catching event		<i>?Y</i>		<i>!Y</i>
Parallel gateway		$\langle + \rangle$		$\langle + \rangle$
Exclusive gateway		$\langle \times \rangle$		$\langle \times \rangle$

Table 5.4: Dual table

For instance the dual model of the one previously seen in Figure 5.5 must receives two messages over *X* and *Z* or send one message over *Y* and then send a message

over W . Hence the dual model have basically the same message behaviour of the main model, but contains tasks with opposite purpose: receive instead send or send instead receive, Figure 5.6.

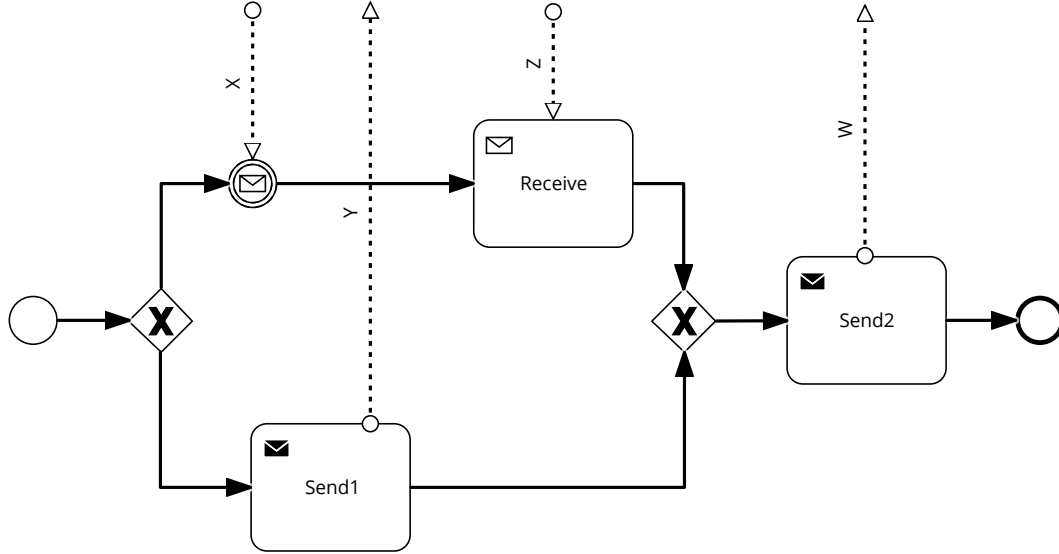


Figure 5.6: The dual model

The dual of a model is not unique, intuitively the presence of a τ task before a message task provides a different model that is always dual. Consequently, due to the absence of τ tasks, it is possible to define the dual of a reduced model as *minimal* dual.

Definition 5.3.1 (Minimal Dual)

Given two BPMN collaboration models M_1 and M_2 , M_2 is the **minimal dual** of M_1 if and only if:

- (i) M_2 is dual to M_1 ;
- (ii) M_2 do not contains τ elements and
- (iii) for all couple of split and join element of M_2 there exists at most one empty path linking them.

With regard to the encoding the dual textual representation can be easily obtained changing sending task with receiving ones, and vice-versa. So that, the dual of encoding in 5.2 is following provided:

$$startEvent \ (?X \ ?Z < \times > !Y) \ !W \ endEvent \quad (5.3)$$

Remembering that only well-structured processes are considered, the model taken into account presents only SESE components. So that the process model can be divided in SESE regions that has a precise behaviour in terms of messages. Any behavioural change in a region may entail an overall change in messages behaviour. However it is possible to modify a region keeping the message behavior unchanged by using non communicative τ tasks also grouped in SESE regions.

The non communicative regions may occur in sequence, in parallel, or exclusively, before, after or in parallel to SESE region².

Adding τ tasks in this way keep untouched the overall messages behaviour. Intuitively, any dual model can be created by adding non communicative regions added to a minimal dual model.

Furthermore any non communicative regions added to the minimal dual model can be deleted using the reduction rules. Hence is it plausible to deduce that any dual model can be reduced to the same minimal dual.

5.3.2 Consistency Checking

Lastly, the possibility to use the encoding also for check consistency is discussed in detail.

The outcome of the previous section show that two model can communicate, hence be consistent, if and only if they are one the dual of the other. So that, in order to check whether or not two models are dual, and consequently consistent, behavioural equivalence techniques may be helpful [24]. However these techniques relies on using formal methods that requires a mapping of BPMN models to Petri nets, labelled transition systems and so on. Moreover, taken a model there exists infinitely many dual models of arbitrary dimension with the same message behaviour. Knowing that, checking behavioural equivalence suffers from the state explosion problem, hence it is necessary to approach to consistency checking in another way.

The approach for checking consistency proposed here is differently based on the previous provided intuitions. In particular, taken two models M_1 and M_2 , first the relative encoding \mathbb{E}_1 and \mathbb{E}_2 are derived. Then, the previously seen reduction technique is applied in order to obtain the reduced encoding $\dot{\mathbb{E}}_1$ and $\dot{\mathbb{E}}_2$.

Considering that the reduction do not change the models message behaviour, M_1 is consistent with M_2 if and only if $\dot{\mathbb{E}}_1$ and $\dot{\mathbb{E}}_2$ represents dual models. Hence $\dot{\mathbb{E}}_1$ and $\dot{\mathbb{E}}_2$ have to be in such way equivalent.

The equivalence between two encoding, $\dot{\mathbb{E}}_1 \equiv \dot{\mathbb{E}}_2$, can be shown considering that the encoding have always the same structure, due to the restriction of elements and topology. In fact, a generic encoding presents tasks or a composition of task between brackets to perform in sequence, one after the other. Compositions of tasks can also be nested one into the other.

The equivalence between two tasks sequences is easily provided by checking if they are lexicographically equals apart from the symbols $!$ that is mapped to $?$ and vice-versa. While two composition of tasks are equivalent components if both are parallel or exclusive compositions, and if any tasks sequence or composition in it, are pairwise equals.

This procedure is now explained using a practical example. Given the two models, M_1 and M_2 , represented respectively in Figures 5.7 and 5.8, the two encoding are following provided:

$$\begin{aligned} \mathbb{E}_1 ::= & \text{startEvent } !X(?Y < \times > \tau \text{ } !Z < \times > (\tau < + > \tau)\tau) \\ & (?W < \times > \tau)\text{endEvent} \end{aligned} \quad (5.4)$$

²The exclusive composition is not taken into account due to the fact that the region can not be performed changing the message behaviour.

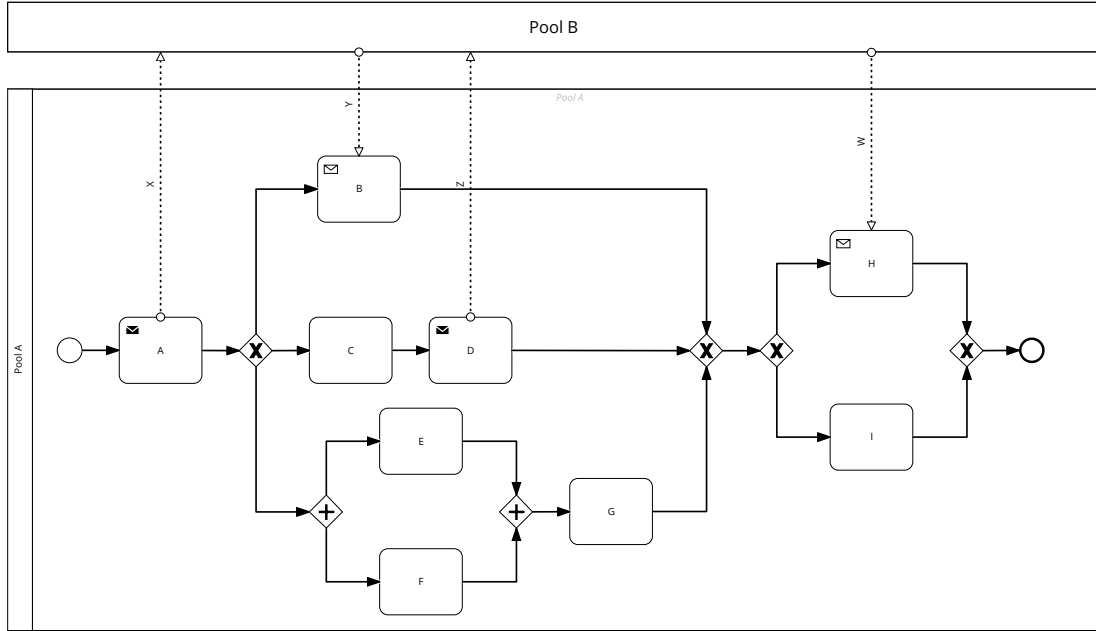


Figure 5.7: M_1 Model

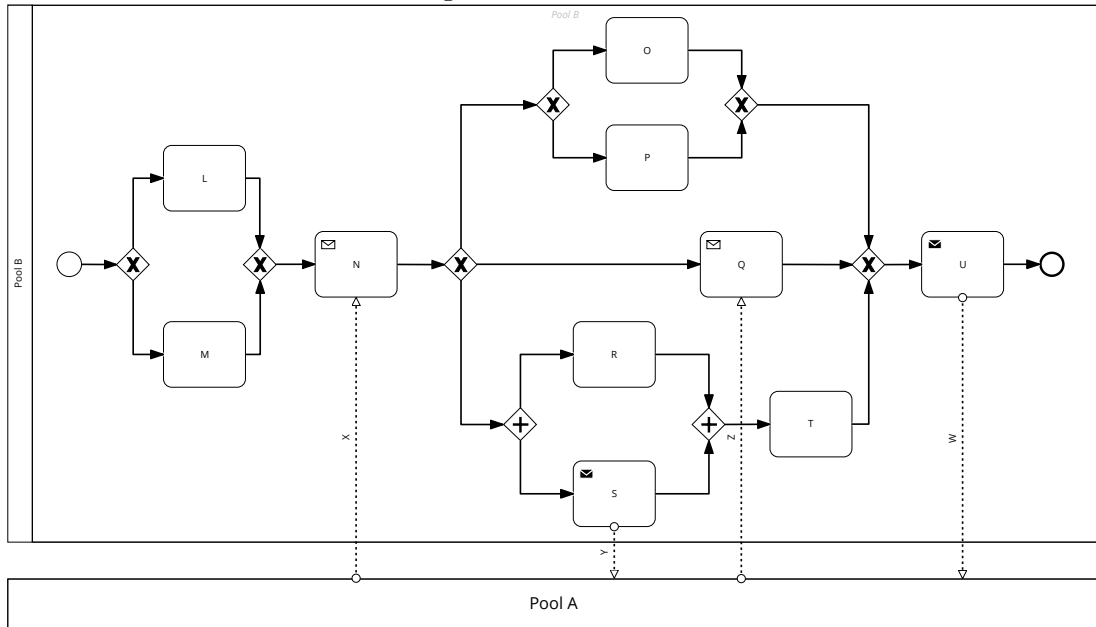


Figure 5.8: M_2 Model

$$\begin{aligned} \mathbb{E}_2 ::= & startEvent(\tau < \times > \tau)?X((\tau < \times > \tau) < \times > ?Z < \times > (\tau < + > !Y)\tau) \\ & !WendEvent \end{aligned} \quad (5.5)$$

The application of reduction rules transform the models discarding all the insignificant task. Figures 5.9 and 5.10 shows the model after the reduction procedure, consequently the reduced encoding become more compact as shown in Equations 5.6 and 5.7.

$$\dot{\mathbb{E}}_1 ::= startEvent \ !X(?Y < \times > !Z)?W \ endEvent \quad (5.6)$$

$$\dot{\mathbb{E}}_2 ::= startEvent \ ?X(?Z < \times > !Y)!WendEvent \quad (5.7)$$

The two models M_1 and M_2 are consistent if and only if $\dot{\mathbb{E}}_1 \equiv \dot{\mathbb{E}}_2$. The equivalence is checked analyzing the sequences of tasks and the compositions of $\dot{\mathbb{E}}_1$ and $\dot{\mathbb{E}}_2$. Both the encoding are composed by a tasks sequence, then a composition and finally another tasks sequence. These three piece of encoding are equivalent, considering the previous defined lexicographical comparison.

In fact, the diagram in Figure 5.11 shows that $\dot{\mathbb{E}}_1$ is composed by three sequences: $startEvent \ !X$, $(?Y < \times > !Z)$ and $?W \ endEvent$. As for $\dot{\mathbb{E}}_2$ that can be divided into $startEvent \ ?X$, $(?Z < \times > !Y)$ and $!W \ endEvent$. Hence $\dot{\mathbb{E}}_1 \equiv \dot{\mathbb{E}}_2$ if this six sequences are pairwise equivalent. The firsts and thirds sequences are lexicographically the same a part from the exclamation and the question marks, so they are equivalent. The two compositions in the middle of the two encoding present the same gateway and have equivalent tasks sequences. Hence, the equivalence of the two reduced encoding implies the consistency between M_1 and M_2 .

$$\begin{aligned} \dot{\mathbb{E}}_1 & \equiv \dot{\mathbb{E}}_2 \\ startEvent \ !X(?Y < \times > !Z)?W \ endEvent & \equiv \\ startEvent \ ?X(?Z < \times > !Y)!WendEvent & \end{aligned} \quad (5.8)$$

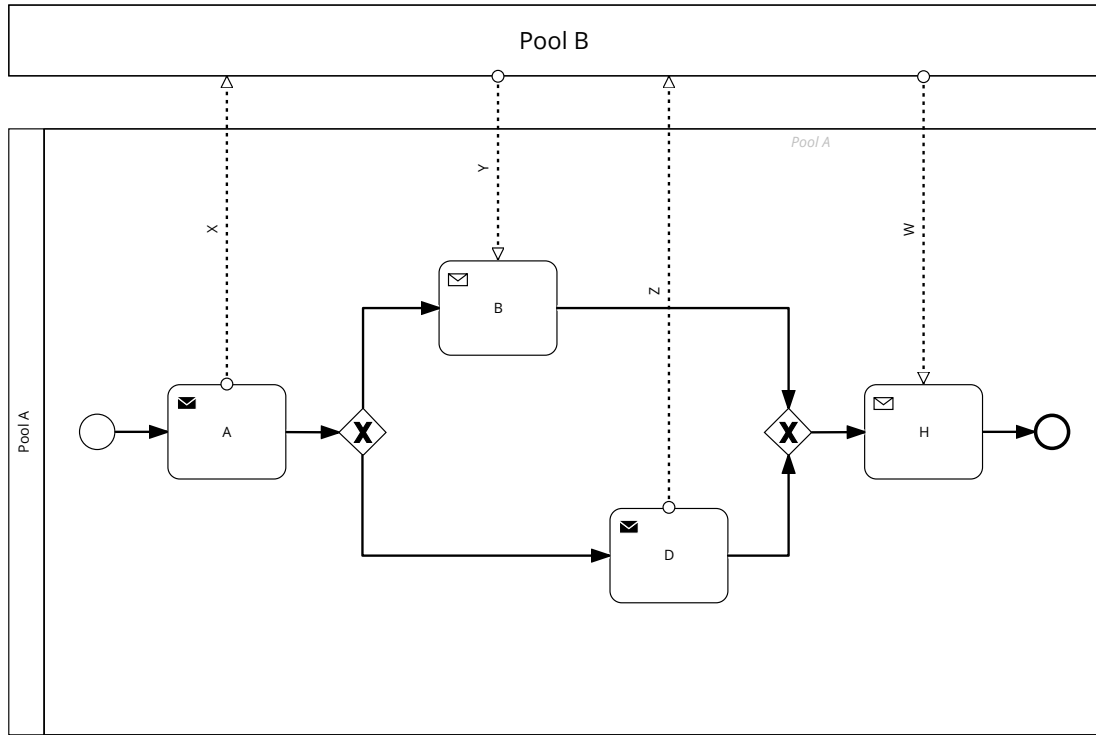


Figure 5.9: Reduced M_1 model

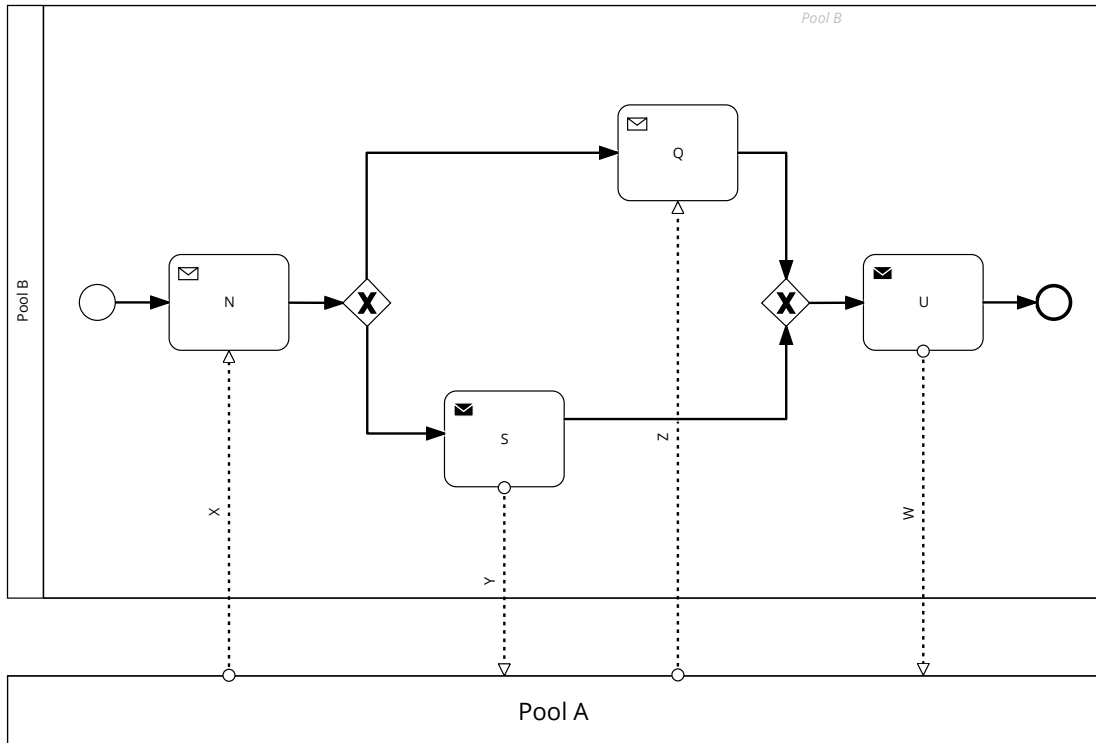
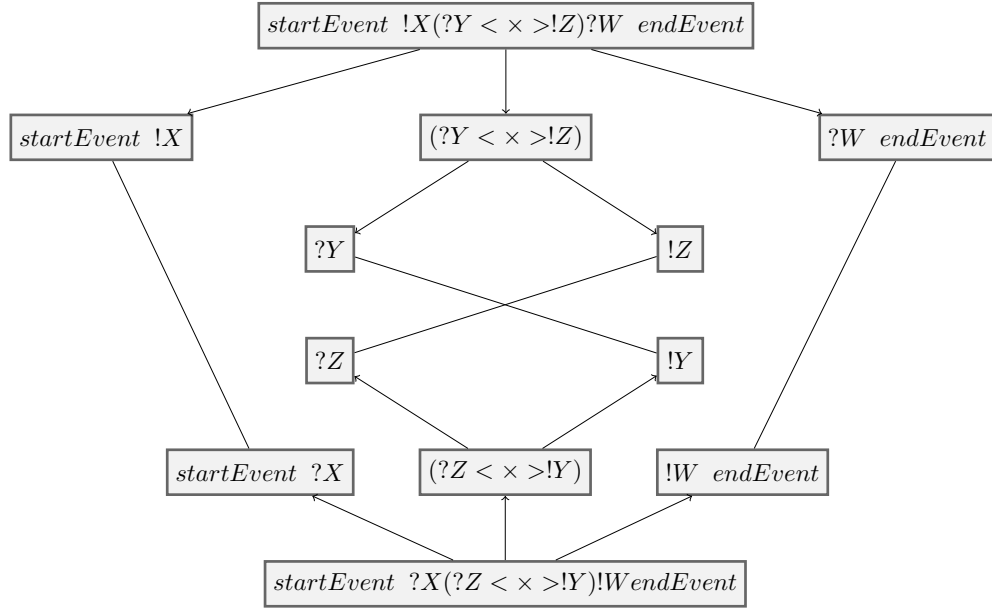


Figure 5.10: Reduced M_2 model


 Figure 5.11: $\dot{\mathbb{E}}_1$ and $\dot{\mathbb{E}}_2$ equivalence

6. Implementation of Consistency Checking Method

This chapter presents a first implementation of the proposed consistency checking method based on BPMN encoding. The described implementation do not aims to provide a fully stable tool, but allows the encoding, the reduction and the consistency checking methods for the considered subset of BPMN collaboration models.

The first section analyzes the overall architecture of the software in order to show the purpose of each tool components. Then in the second and third sections are presented more in detail the tool classes.

6.1 Architectural Overview

The proposed tool is based on *Java*[32] programming language and it is integrated into the *Eclipse* BPMN modeling environment¹. The tool allows several uses:

- translation from/to BPMN xml;
- encoding generation;
- encoding reduction;
- dual encoding and model generation;
- consistency checking.

More in detail, the tool is composed by two packages, as shown in the class diagram in Figure 6.1. The first package, named *consistency*, manages the model graph in order to generate the reduced model, the dual model and check consistency. While the *parsers* one is responsible of the translation, in both direction, of *xml* to encoding strings array and implements consequently the elements for the model graph and set the positioning of the elements for the *xml* creation.

In order to provide a discussion on each class purpose, it is necessary to introduce the data structures used to handle the model in input and the resulting encoding.

The tool memorizes the BPMN model with graph as data structure. The graph implementation is taken from the open source *Java* library *jGraphT*[25]. The model graph, Figure 6.2 is formally a directed graph in which the vertex elements correspond to the set *El* of model elements: tasks, events and gateways. While the edge set correspond to the model sequence flows set. Hence the model graph has the same

¹<http://www.eclipse.org/bpmn2-modeler/>

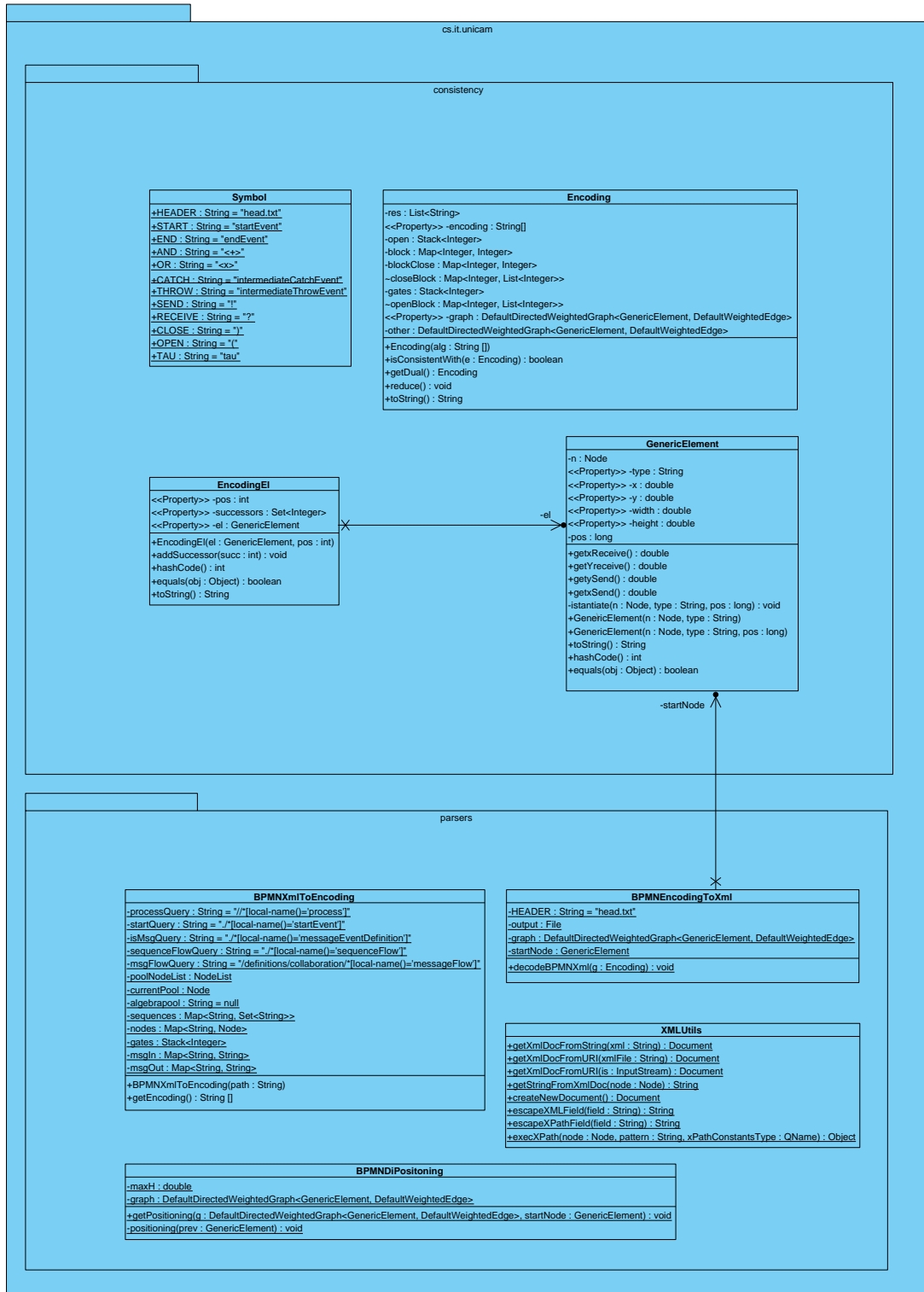


Figure 6.1: Class diagram

topology of the process model. Each element of the model is implemented by a custom class named *GenericElement* that contains the node information such as the type of element and its xml parameters.

The textual representation of the encoding is provided as a string array, *Strng[]*. At

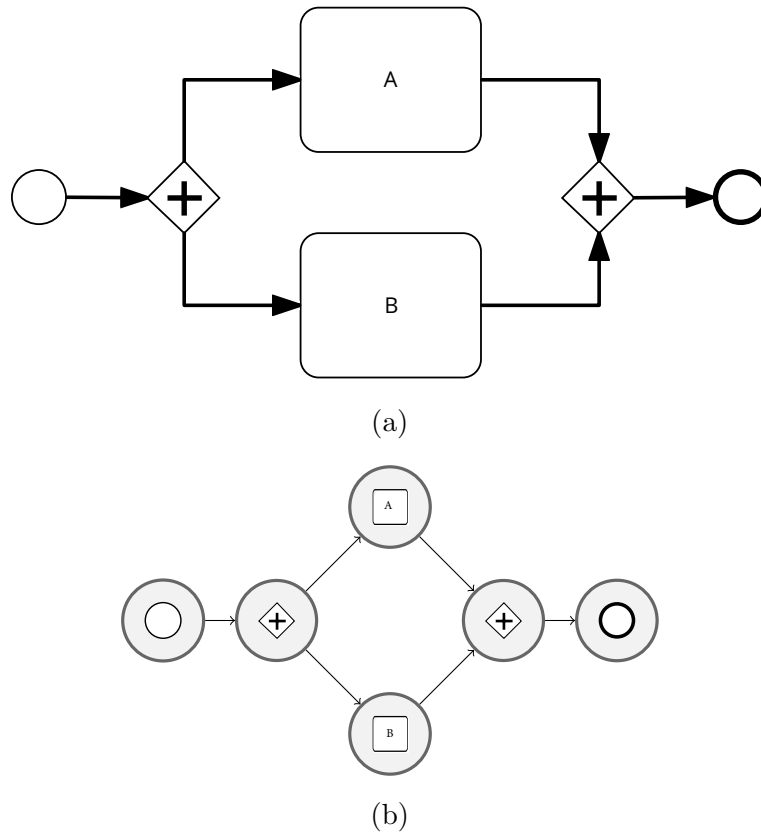


Figure 6.2: BPMN model example (a) and its model graph (b)

each position of the array correspond to a task, a event, a gateway or a bracket (see Figure 6.3).

startEvent	!X	(τ	< \times >	!Y)	endEvent
0	1	2	3	4	5	6	7

Figure 6.3: Encoding array example

Hence, are now introduced the most relevant tool classes in the packages, considering their purposes and illustrating their algorithms.

6.2 BPMN Model Parsing

The parsers package contains all the classes needed to acquire a BPMN model as a *xml* file and transform it into the encoding, and vice-versa. The classes contained in this package are following listed:

- **BPMNEncodingToXml**, translates an encoding to its *xml* model;
- **BPMNDiPositioning**, assigns to each BP element its position inside the model;
- **BPMNXmlToEncoding**, translates an *xml* model to its encoding;
- **XMLUtils**, allows to query the *xml* input file.

The first class of this package, that I am going to introduce, is *BPMNXmlToEncoding*. This class implements the methods useful to convert a *xml* BPMN model into an *Encoding* object. The class constructor takes as input the *xml* file path in order to instantiate the *Document* object².

The *xml* document is now parsed in order to populate the data structures *sequences* with the sequence flows, *nodes* with the model elements, *msgIn* and *msgOut* with the message flows. Moreover the *String encoding* and the *Stack gates* are created.

Once instantiated, the *BPMNXmlToEncoding* object provides the public method *getEncoding()* that returns the encoding string array of the model. The procedure that creates the encoding uses the private methods in Listing 6.1 as follows. From the *nodes* list is peeked the start node, then the method *recursiveParsing(startNode)* is called.

```
1 void recursiveParsing(Node previous){
2     switch (previous.getNodeName()) {
3         case Symbol.START:
4             simpleMove(Symbol.START, previous);
5             break;
6         case Symbol.END:
7             encoding += Symbol.END;
8             break;
9         case "exclusiveGateway":
10            gatewayMove(Symbol.OR, previous);
11            return;
12        case "parallelGateway":
13            gatewayMove(Symbol.AND, previous);
14            return;
15        case "receiveTask":
16            simpleMove(Symbol.RECEIVE + msgIn.get(previous), previous);
17            return;
18        case "sendTask":
19            simpleMove(Symbol.SEND + msgOut.get(previous), previous);
20            return;
21        case "intermediateCatchEvent":
22            simpleMove(Symbol.RECEIVE + msgIn.get(previous), previous);
23            return;
24        case "intermediateThrowEvent":
25            simpleMove(Symbol.SEND + msgOut.get(previous), previous);
26            return;
27        default:
28            simpleMove(Symbol.TAU, previous);
29            return;
30    }
31 }
32
33 void gatewayMove(String type, Node previous) throws Exception {
34     if (sequences.get(previous).size() > 1) {
35         gates.push(sequences.get(previous).size());
36         int count = 0;
37         for (String id : sequences.get(previous)) {
38             if (count > 0) {
39                 encoding += type + " ";
40             } else {
41                 encoding += Symbol.OPEN+" ";
```

²org.w3c.dom.Document

```

42     }
43     count++;
44     recursiveParsing(nodes.get(id));
45 }
46 } else {
47     int openBranch = gates.pop();
48     if (openBranch > 1) {
49         gates.push(openBranch - 1);
50         return;
51     } else {
52         encoding += Symbol.CLOSE+" ";
53         for (String id : sequences.get(previous)) {
54             recursiveParsing(nodes.get(id));
55             break;
56         }
57     }
58 }
59 }
60
61 void simpleMove(String type, Node previous) throws Exception {
62     encoding += type + " ";
63     for (String id : sequences.get(previous)) {
64         recursiveParsing(nodes.get(id));
65         break;
66     }
67 }

```

Listing 6.1: Xml to Encoding translation

Recursively, the method identify the input node type. Consequently, it call the method *gatewayMove* if the node type is a gateway, stop the method if the node type is an end event or perform the *simpleMove* otherwise. The *simpleMove* requires the node type adding it to the encoding string and then call the *recursiveParsing* for the following model element. Differently, *gatewayMove* method distinguish the split from the join gateway checking the successors quantity. The stack *gates* is used to keep the branching information: whenever a split gateway occurs, the outgoing flows number is pushed in the stack. Hence, once a join gateway occurs, the number in the head of the stack corresponds to its number of incoming sequences. This information allows to add into the encoding the gateway symbols both with its brackets.

The parsing phase perform queries in the *xml* structure using the methods provided in *XmlUtils*³.

Intuitively the *BPMNEncodingToXml* fulfil the opposite purpose, it takes an Encoding object and creates in the tool path a new BPMN xml file of the encoded model. The public method *decodeBPMNXml* navigates the model graph from the start event until the end one is reached. Each element, sequence flow or message flow is translated to xml structure. Moreover the class calls the *BPMNDiPositioning* in order to assign the graphical information of the model, i.e. element sizes, space positioning.

³This class is realized by Damiano Falcioni - Unicam - damiano.falcioni@gmail.com

6.3 Consistency Checking

The consistency package is the core of the tool. It provides the classes useful for create and manipulate the Encoding object. The package is composed by the following classes:

- **Encoding**, manages the encoding and provides consistency checking method;
- **EncodingEl**, characterizes the elements of the encoding;
- **GenericElement**, represents the model graph vertices;
- **Symbol**, collects the string constants used by the other classes.

In this package the class *Symbol* keeps in memory, as string, constants used by each package classes. However, the main class of this package is the *Encoding* one, it manipulate the encoding array in order to apply the reduction, obtain the dual and check consistency. It can be instantiated with a *String[]* object resulting from the parsing of an *xlm* or an arbitrary encoding array. The created object allows several operations: dual creation, reduction and consistency checking.

```

1 Encoding getDual() {
2     String[] x = encoding;
3     for (int i = 0; i < encoding.length; i++) {
4         if (encoding[i].contains(Symbol.SEND)) {
5             x[i] = encoding[i].replace(Symbol.SEND, Symbol.RECEIVE);
6         } else if (encoding[i].contains(Symbol.RECEIVE)) {
7             x[i] = encoding[i].replace(Symbol.RECEIVE, Symbol.SEND);
8         }
9     }
10    return new Encoding(x);
11 }

```

Listing 6.2: getDual method

The method *getDual* returns the dual of the object by replacing the question marks with the exclamation ones and vice-versa (Listing 6.2). While the method *reduce* first delete from the model graph each τ element applying the rule **Tau**. Then, due to the fact that the graph used for the model discard equals edges, the elimination of τ elements, in the parallel or exclusive branches, implies the presence of patterns in which only **And_1**, **And_2**, **Xor_1** and **Xor_2** can be applied. Lastly, the previously seen recursive parsing modify the encoding string array.

```

1 void reduce() {
2     for (GenericElement e : graph.vertexSet()) {
3         if (e.getType().equals(Symbol.TAU)) {
4             toRemove.add(e);
5             for (DefaultWeightedEdge s : graph.incomingEdgesOf(e)) {
6                 for (DefaultWeightedEdge t : graph.outgoingEdgesOf(e)) {
7                     graph.addEdge(graph.getEdgeSource(s), graph.getEdgeTarget(
8                         t), new DefaultWeightedEdge());
9                 }
10            }
11        }
12    }
13    graph.removeAllVertices(toRemove);
14    toRemove.clear();

```

```

14  for (GenericElement e : graph.vertexSet()) {
15      if (e.getType().equals(Symbol.AND)) {
16          if (graph.outDegreeOf(e) > 1) {
17              GenericElement last = null, middle = null;
18              for (DefaultWeightedEdge s : graph.outgoingEdgesOf(e)) {
19                  if (graph.getEdgeTarget(s).getType().equals(Symbol.AND)) {
20                      edgeToRemove.add(s);
21                      last = graph.getEdgeTarget(s);
22                  } else {
23                      middle = graph.getEdgeTarget(s);
24                  }
25              }
26              if (graph.outDegreeOf(e) == 2) {
27                  for (DefaultWeightedEdge k : graph.outgoingEdgesOf(last))
28              {
29                  sourceToAdd.add(middle);
30                  targetToAdd.add(graph.getEdgeTarget(k));
31              }
32              for (DefaultWeightedEdge k : graph.incomingEdgesOf(e)) {
33                  sourceToAdd.add(graph.getEdgeSource(k));
34                  targetToAdd.add(middle);
35              }
36              toRemove.add(e);
37              toRemove.add(last);
38          } else if (graph.outDegreeOf(e) == 1) {
39              for (DefaultWeightedEdge s : graph.outgoingEdgesOf(e)) {
40                  GenericElement node = graph.getEdgeTarget(s);
41                  if (node.getType().equals(Symbol.AND)) {
42                      if (graph.outDegreeOf(node) == 1) {
43                          for (DefaultWeightedEdge d : graph.outgoingEdgesOf(
44                          node)) {
45                              GenericElement last = graph.getEdgeTarget(d);
46                              if (last.getType().equals(Symbol.AND)) {
47                                  edgeToRemove.add(s);
48                                  edgeToRemove.add(d);
49                                  toRemove.add(node);
50                                  toRemove.add(last);
51                                  for (DefaultWeightedEdge k : graph.
52                                  outgoingEdgesOf(last)) {
53                                      sourceToAdd.add(e);
54                                      targetToAdd.add(graph.getEdgeTarget(k));
55                                  }
56                              }
57                          }
58                      }
59                  }
60              } else if (e.getType().equals(Symbol.OR)) {
61                  if (graph.outDegreeOf(e) == 1) {
62                      for (DefaultWeightedEdge s : graph.outgoingEdgesOf(e)) {
63                          GenericElement node = graph.getEdgeTarget(s);
64                          if (node.getType().equals(Symbol.OR)) {
65                              if (graph.outDegreeOf(node) == 1) {
66                                  for (DefaultWeightedEdge d : graph.outgoingEdgesOf(

```

```
node)) {
67         GenericElement last = graph.getEdgeTarget(d);
68         if (last.getType().equals(Symbol.OR)) {
69             edgeToRemove.add(s);
70             edgeToRemove.add(d);
71             toRemove.add(node);
72             toRemove.add(last);
73             for(DefaultWeightedEdge k : graph.outgoingEdgesOf(
last)){
74                 sourceToAdd.add(e);
75                 targetToAdd.add(graph.getEdgeTarget(k));
76             }
77         }
78     }
79 }
80 }
81 }
82 } else if (graph.outDegreeOf(e) > 1){
83     boolean done = false;
84     for (DefaultWeightedEdge s : graph.outgoingEdgesOf(e)) {
85         GenericElement node = graph.getEdgeTarget(s);
86         if (node.getType().equals(Symbol.OR)) {
87             if(!done){
88                 GenericElement tmp = new GenericElement(null, Symbol.
TAU);
89                 toAdd.add(tmp);
90                 sourceToAdd.add(e);
91                 targetToAdd.add(tmp);
92                 sourceToAdd.add(tmp);
93                 targetToAdd.add(node);
94             }
95             edgeToRemove.add(s);
96             done = true;
97         }
98     }
99 }
100 }
101 }
102 }
103 graph.removeAllVertices(toRemove);
104 graph.removeAllEdges(edgeToRemove);
105 for (GenericElement v : toAdd) {
106     graph.addVertex(v);
107 }
108 for (int i = 0; i < sourceToAdd.size(); i++) {
109     graph.addEdge(sourceToAdd.get(i), targetToAdd.get(i));
110 }
111 this.encoding = recursiveParsing(start);
112 }
```

Listing 6.3: reduce method

The *Encoding* class contains also a method for checking consistency. The method checks consistency over the model graphs instead the encoding string array, as seen theoretically before. This choice comes from the architectural approach in which the model graph is used to keep in memory the model messages protocol. Hence the method

isConsistentWith tells whether or not the considered encoding and the input one are dual, comparing the reduced model graphs.

More in details, *graph* refers to the reduced model graph of the considered encoding, while *other* refers to the input reduced model graph. The method returns true if and only if *graph* and *other*, from the start event nodes, are able to reach nodes of the same type or, if the reached vertex refers to a communicating element, the nodes communicates in the same channel but in opposite direction.

In order to simplify this procedure, the *other* encoding graph is created using its dual. Hence every reachable nodes have to be of the same type also for the communicative elements. First a check on the order and the size of the two graph is performed due to the fact that this quantity have to be the same in both graphs, i.e. both graphs have the same number of edges and nodes. Consequently, the method *checkFlow* recursively checks the couples of reachable nodes starting from the start event, until the end one is reached. If the method reach the end node in both graphs, then the models are consistent each other.

```

1 public boolean isConsistentWith(Encoding e) {
2     if(e==null) {
3         return false;
4     }
5     reduce();
6     e.reduce();
7     e = e.getDual();
8     other = e.getGraph();
9     graph = getGraph();
10    if (other.vertexSet().size() != other.vertexSet().size() || other.
        edgeSet().size() != other.edgeSet().size()) {
11        return false;
12    }
13    return checkFlow(graph.getStart(), other.getStart());
14 }
15
16 boolean checkFlow(GenericElement s1, GenericElement s2) {
17     if (graph.outDegreeOf(s1) != other.outDegreeOf(s2)) {
18         return false;
19     } else if (graph.outDegreeOf(s1) == 0) {
20         return true;
21     }
22     Set<GenericElement> t1 = new HashSet<GenericElement>();
23     Set<GenericElement> t2 = new HashSet<GenericElement>();
24     for (DefaultWeightedEdge e1 : graph.outgoingEdgesOf(s1)) {
25         t1.add(graph.getEdgeTarget(e1));
26     }
27     for (DefaultWeightedEdge e2 : other.outgoingEdgesOf(s2)) {
28         t2.add(graph.getEdgeTarget(e2));
29     }
30     Map<GenericElement, GenericElement> toCheck = new HashMap<
        GenericElement, GenericElement>();
31     GenericElement del = null;
32     boolean ret = false;
33     for (GenericElement e1 : t1) {
34         check = false;
35         for (GenericElement e2 : t2) {
36             if (e1.isDual(e2)) {

```

```
37         toCheck.put (e1, e2);
38         del = e2;
39         check = true;
40         break;
41     }
42 }
43 if (!check) {
44     return false;
45 }
46 t2.remove (del);
47 }
48 ret = true;
49 for (GenericElement x1 : toCheck.keySet()) {
50     ret &= checkBisim(x1, toCheck.get(x1));
51 }
52 return ret;
53 }
```

Listing 6.4: Consistency checking method

7. Validation

This chapter provides a practical usage of the presented tool in order to validate our results. In the first section the study case is presented using the previously seen modeling approaches. Hence, the second section shows the tool functionality validating our consistency checking method.

7.1 Order Fulfillment Validation Scenario

In this section is presented, more in detail, the case study, introduced previously, regarding an order fulfillment. Briefly, the case study model is a BPMN collaboration with two participants: the seller and the supplier. In order to complete the order, the seller have to check the raw materials availability. Whenever the raw materials are not in stock, the seller request the material to the supplier that send the requested goods. Then the seller perform the tasks for the shipment and payment concurrently. Finally the process ends after the order archiving. The raw material acquisition and the shipment and payment can be modelled in the overall process as a sub-processes. Given that only the first one communicates with the supplier, the material acquisition sub-process is taken into account for our purpose.

Hence, the order fulfillment business process could be modeled following the introduced six approaches of represent the sub-process¹. The models derived from the approaches present two communicating participants, the seller and the supplier, that are modeled collapsed or expanded using single or multilayer model. In order to present this six approach, Figure 7.1 present the abstract layer of the model in which the *Supplier* (**SUPP**) pool is expanded, hence its behaviour is represented. While *Acquire raw material* (**ARM**) sub-process representation is provided in the detailed layer, shown in Figure 7.2.

The approaches are hence the following:

- expanded **ARM** sub-process and collapsed **SUPP** pool;
- expanded **ARM** sub-process and expanded **SUPP** pool;
- collapsed **ARM** sub-process and collapsed **SUPP** pool;
- collapsed **ARM** sub-process and expanded **SUPP** pool;
- Multilayer collapsed **ARM** sub-process and collapsed **SUPP** pool;
- Multilayer collapsed **ARM** sub-process and expanded **SUPP** pool.

¹The *Shipment and invoice* sub-process is represented collapsed in order to make the model smaller.

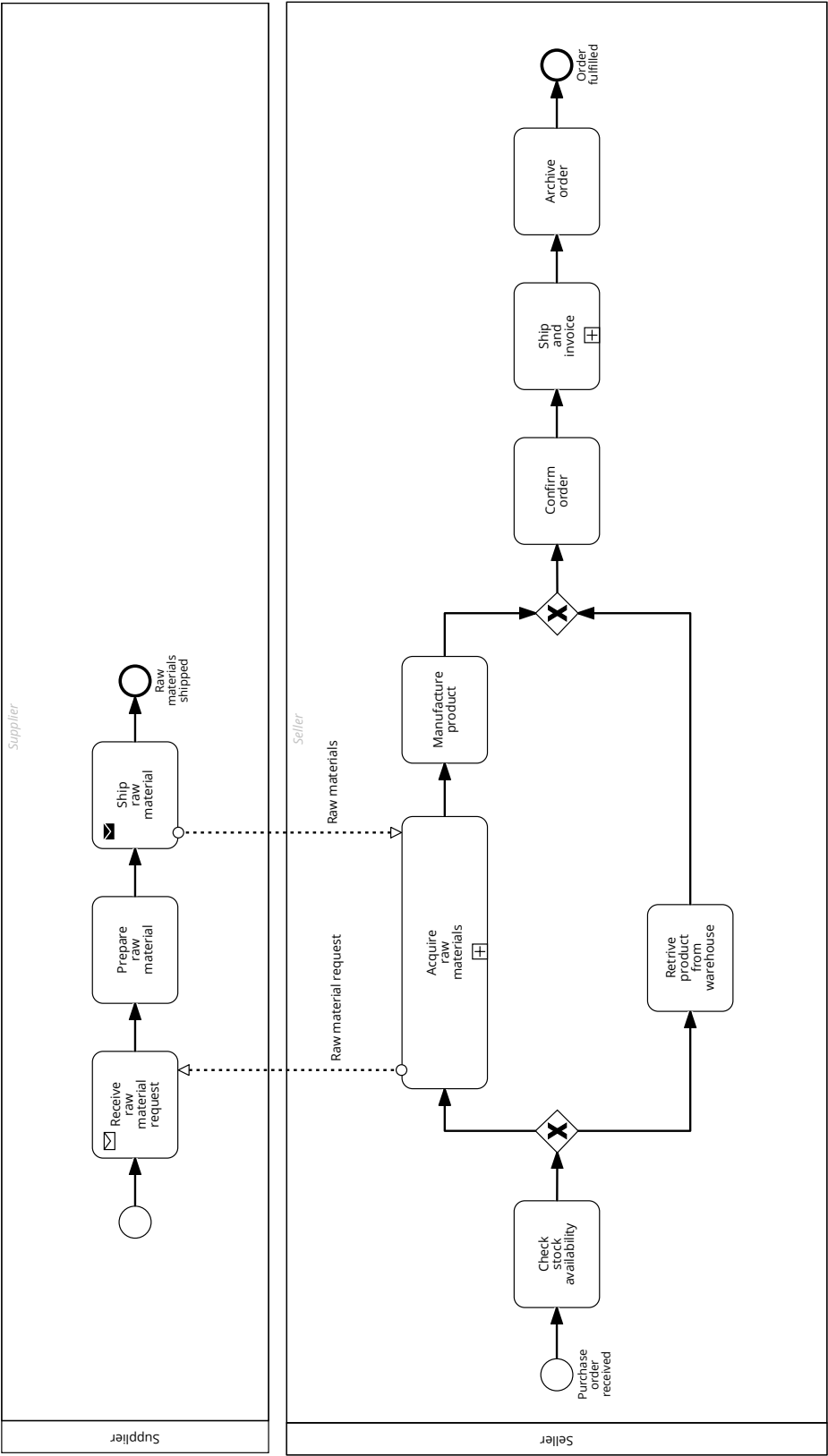
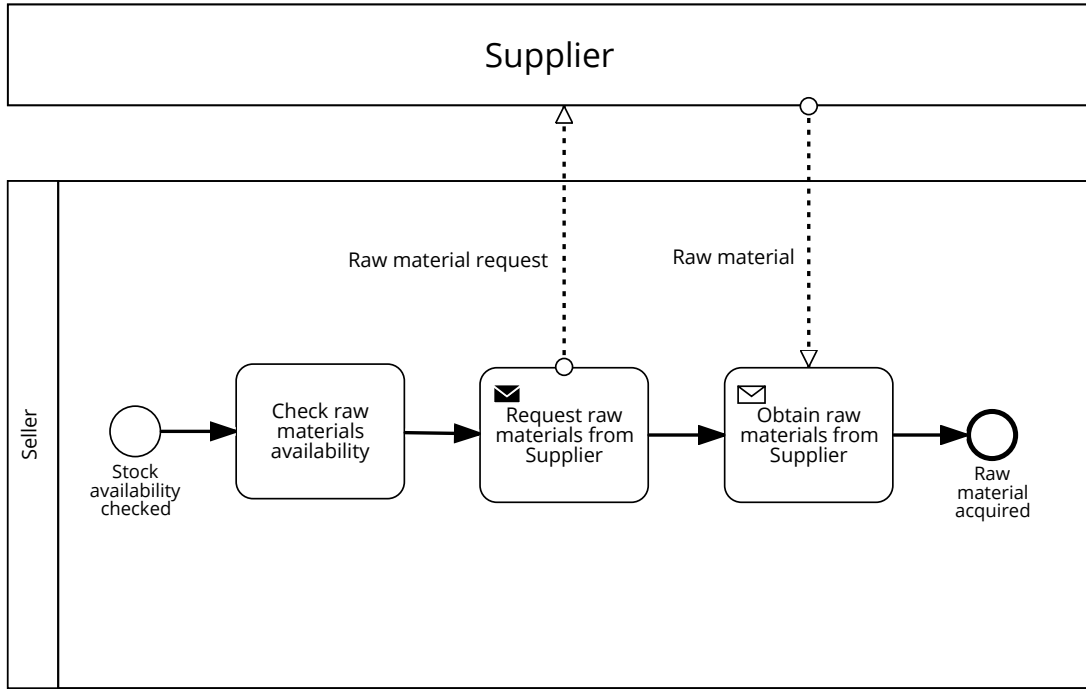


Figure 7.1: Case Study: abstract layer with collapsed sub-processes and expanded pools

Figure 7.2: Case Study: *Acquire raw materials* detailed layer

According to the approaches comparison table previously introduced, Table 7.1 resumes what the presented tool is able to obtain. In detail, giving as input to the tool the **ARM** sub-process model, it have to provides a consistent model for **SUPP** and vice-versa. Moreover, if both **ARM** and **SUPP** are provided explicitly, the tool have to answer it they are or not consistent.

			Participants		Consistency
			ARM	SUPP	
Approaches	Exp. ARM	Coll. SUPP	P	D	
		Exp. SUPP	P	P	V
	Coll. ARM	Coll. SUPP			
		Exp. SUPP	D	P	
	M Coll. ARM	Coll. SUPP	P	D	
		Exp. SUPP	P	P	V

Table 7.1: Approaches expectation table

7.2 Consistency Checking via Tool Application

The analysis done for the case study requires from the tool the fulfillment of three tasks:

- (i) derive from **ARM** model a consistent one describing **SUPP**;
- (ii) derive from **SUPP** model a consistent one describing **ARM**;
- (iii) check consistency between **ARM** and **SUPP** models.

In order to achieve this goal a *Test* class, shown in Listing 7.1, is provided. This class implements the toll methods in order to firstly load the models in *xml* format of **ARM** and **SUPP** models. Then, the class allows to check the tool purposes. First the models are parsed and consequently their encoding are provided and printed in the output console. Then the method *reduce()* change **ARM** and **SUPP** models applying reduction rules. The resulting reduced encoding are printed in the output console.

Consequently, both **ARM** and **SUPP** reduced encoding are used to derive their minimal dual using the method *getDual()*. The resulting encoding are printed in the output console and are parsed in order to obtain the *xml* representations. Hence, two *xml* model are provided as output parsing the **ARM** and **SUPP** minimal dual. This files are saved in the same path of the tool with a unique name. Finally the consistency checking method, *isConsistentWith()* is called in order to answer if *ARM* and *SUPP* are or not consistent.

```
1 public class Test {
2     public static void main(String[] args) throws Exception {
3         // Parsing of ARM model to Encoding
4         BPMNXmlToEncoding arm = new BPMNXmlToEncoding("arm.bpmn");
5         // Parsing of SUPP model to Encoding
6         BPMNXmlToEncoding supp = new BPMNXmlToEncoding("supp.bpmn");
7         // ARM encoding creation
8         Encoding armEnc = new Encoding(arm.getEncoding());
9         System.out.println("Encoding of ARM model: " + armEnc);
10        // SUPP encoding creation
11        Encoding suppEnc = new Encoding(supp.getEncoding());
12        System.out.println("Encoding of model SUPP model: " + suppEnc);
13
14        //(i) Derive from ARM model a consistent one describing SUPP
15        // ARM model reduction
16        armEnc.reduce();
17        System.out.println("Reduced encoding of ARM model: " + armEnc);
18        // ARM dual
19        Encoding armMDual = armEnc.getDual();
20        System.out.println("Dual of ARM model: " + armMDual);
21        // Parsing of ARM minimal dual to XML
22        BPMNEncodingToXml.decodeBPMNXml(armMDual);
23
24        //(ii) Derive from SUPP model a consistent one describing ARM
25        // SUPP model reduction
26        suppEnc.reduce();
27        System.out.println("Reduced encoding of SUPP model: " + suppEnc);
28        ;
29        // SUPP dual
30        Encoding suppMDual = suppEnc.getDual();
31        System.out.println("Dual of SUPP model: " + suppMDual);
32        // Parsing of SUPP minimal dual to XML
33        BPMNEncodingToXml.decodeBPMNXml(suppMDual);
34
35        //(iii) Consistency checking between ARM and SUPP
36        System.out.println("ARM is consistent with SUPP?: " + armEnc.
37        isConsistentWith(suppEnc));
38    }
39 }
```

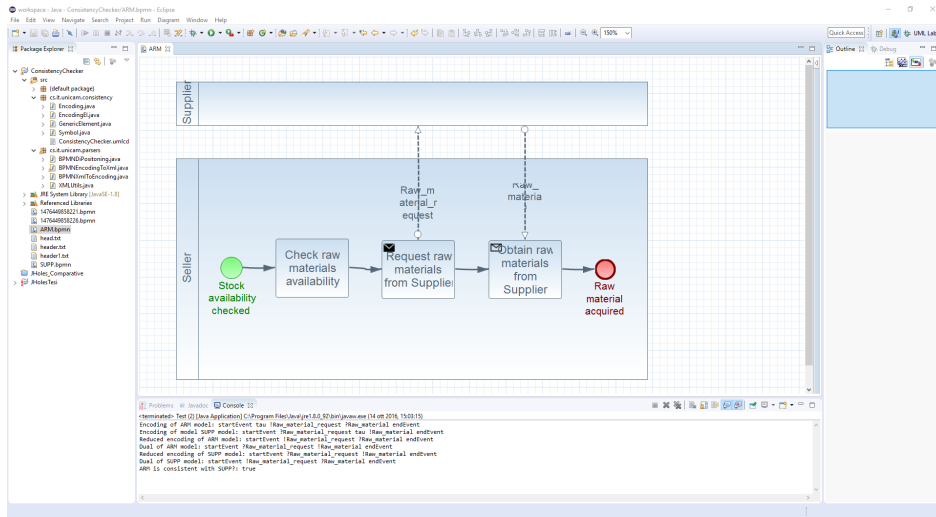
Listing 7.1: Running example main method

Figure 7.3 shows the output provided by the console after the run of *main* method in *Test* class.

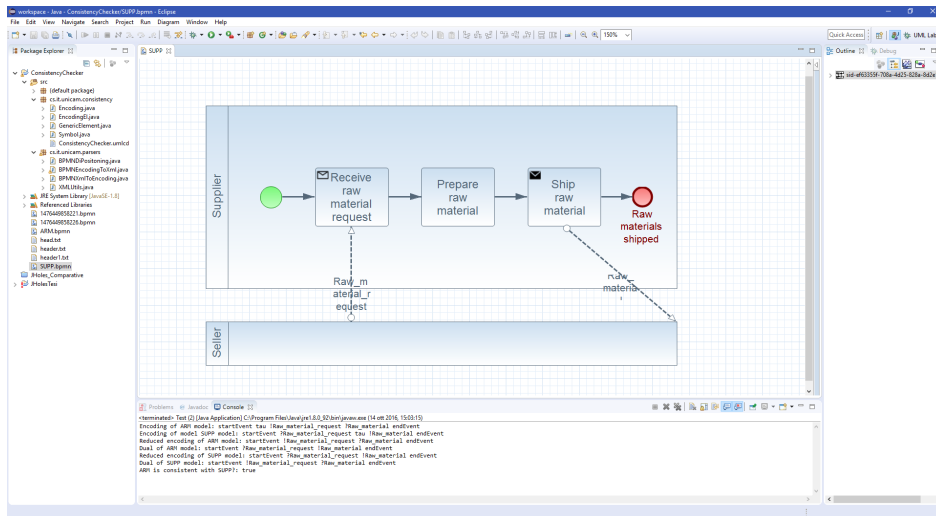
```
<terminated> Test (2) [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (11 ott 2016, 17:38:06)
Encoding of ARM model: startEvent tau !Raw_material_request ?Raw_material endEvent
Encoding of SUPP model: startEvent ?Raw_material_request tau !Raw_material endEvent
Reduced encoding of ARM model: startEvent !Raw_material_request ?Raw_material endEvent
Dual of ARM model: startEvent ?Raw_material_request !Raw_material endEvent
Reduced encoding of SUPP model: startEvent ?Raw_material_request !Raw_material endEvent
Dual of SUPP model: startEvent !Raw_material_request ?Raw_material endEvent
ARM is consistent with SUPP?: true
```

Figure 7.3: Console output

The first and the second lines in console output provide the encoding of **ARM** and **SUPP**, this encoding are following shown both with the piece of model that represent:



$ARM := startEvent \tau !Raw_material_request ?Raw_material endEvent$



$SUPP := startEvent ?Raw_material_request \tau !Raw_material endEvent$

Lines four and five and lines six and seven show respectively the reduced encoding and the minimal dual for **ARM** and **SUPP** models.

$$\dot{\mathbf{ARM}} := startEvent !Raw_material_request ?Raw_material endEvent$$

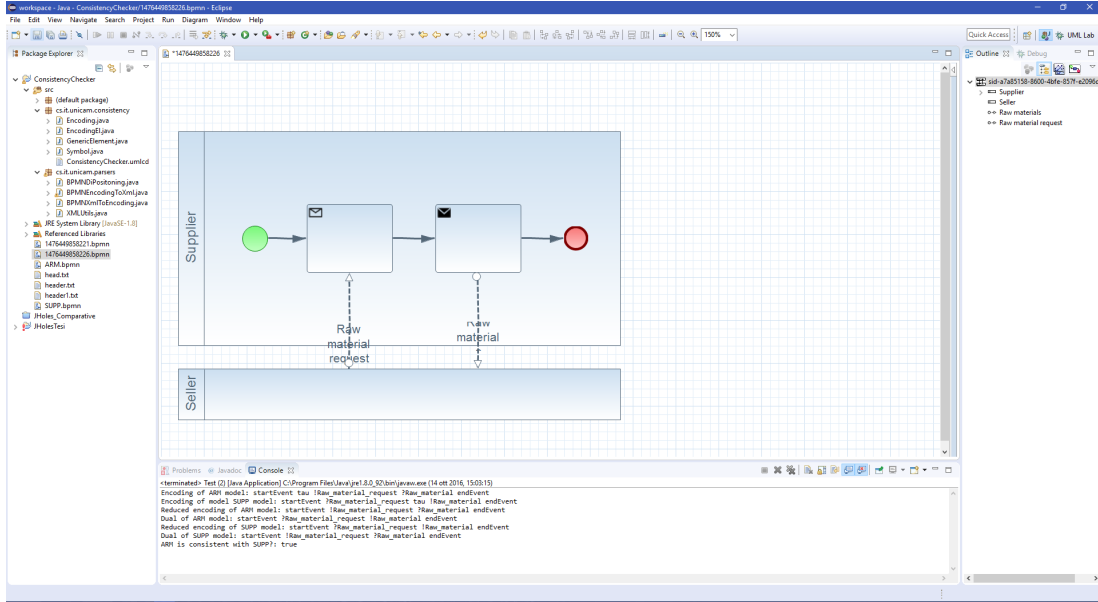
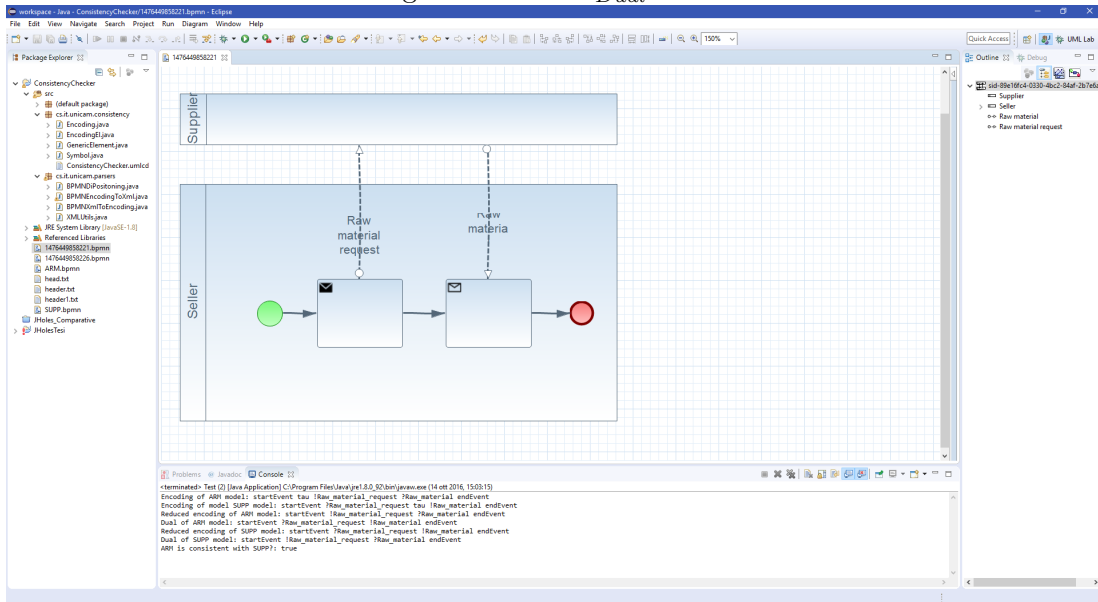
$$\dot{\mathbf{ARM}}_{Dual} := startEvent ?Raw_material_request !Raw_material endEvent$$

$$\dot{\mathbf{SUPP}} := startEvent ?Raw_material_request !Raw_material endEvent$$

$$\dot{\mathbf{SUPP}}_{Dual} := startEvent !Raw_material_request ?Raw_material endEvent$$

Intuitively, it is possible to see that $\dot{\mathbf{ARM}} \equiv \dot{\mathbf{SUPP}}_{Dual}$ and $\dot{\mathbf{SUPP}} \equiv \dot{\mathbf{ARM}}_{Dual}$. However, the last line provided by the output console says that the two piece of models are consistent each other. In fact, once the seller request raw material using an outgoing message flow, named *Raw material request*, the supplier receive the same message flow with a receive task. Then the supplier after a non communicative task mapped with τ , send back to the supplier the requested material using a message flow named *Raw material* that is received by supplier. The communication fit in both side, hence the models are consistent.

Finally, the minimal dual models of **ARM** and **SUPP** provided in output are parsed from the *xml* representation to the graphical one using the *Signavio* BPMN modeler [13]. The resulting models are following provided in Figure 7.4 and 7.5.

Figure 7.4: ARM_{Dual} modelFigure 7.5: $SUPP_{Dual}$ model

Conclusions and Further Development

I faced-up the problem regarding the multilayer consistency checking in BPMN collaboration models that include sub-processes exchanging messages.

I started analysing the domain of the problem and the current state of the art. Unfortunately, in literature, there are no specific contributions regarding consistency of messages protocol in multilayer BPMN collaboration. Hence, a novel consistency checking method based on messages protocol encoding for well-structured BPMN collaboration models is developed.

The proposed messages protocol encoding is a textual representation of messages exchange behaviour that is simplified using a reduction procedure based on reduction rules. This procedure discard all non communicative activities within the model minimizing the textual representation of the message protocol both with the complexity of consistency checking method.

Then I introduce the concept of minimal dual model as the smallest dual model of the messages protocol. The minimal dual result essential in order to check consistency. In fact, is possible to show that two models are consistent if their reduced encoding are equivalent under our equivalence definition that is based on the minimal dual.

Consequently, I developed a tool prototype that implements all this features: encoding from/to *xml*, reduction, dual creation and consistency checking. Our hypothesis are finally validated using the provided tool in a case study.

As further goal I want to formalize the results derived from the research in order to give a rigorous interpretation of consistency checking in BPMN collaboration. In particular, we want to investigate in this regard considering a wider set of BPMN structures such as multiple instance and looped behaviours. About the tool development, the further goals regard the possibility to integrate the protocol encoding within the model, increasing its the understandability.

Bibliography

- [1] Bonnie Brinton Anderson, James V Hansen, Paul Benjamin Lowry, and Scott L Summers. Model checking for design and assurance of e-business processes. *Decision Support Systems*, 39(3):333–344, 2005.
- [2] Farhad Arbab, Natallia Kokash, and Sun Meng. Towards using reo for compliance-aware business process modeling. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 108–123. Springer, 2008.
- [3] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using bpmn-q and temporal logic. In *International Conference on Business Process Management*, pages 326–341. Springer, 2008.
- [4] Michaela Baumann, Michael Heinrich Baumann, and Stefan Jablonski. On behavioral process model similarity matching: A centroid-based approach. 2015.
- [5] Egon Börger and Bernhard Thalheim. A Method for Verifiable and Validatable Business Process Modeling. In *Advances in Software Engineering*, volume 5316, pages 59–115. Springer, Berlin, Heidelberg, 2008. ISBN 978-3-540-89761-3 978-3-540-89762-0. URL http://link.springer.com/10.1007/978-3-540-89762-0_3.
- [6] Andrea Bracciali, Antonio Brogi, and Carlos Canal. Adapting components with mismatching behaviours. In *International Working Conference on Component Deployment*, pages 185–199. Springer, 2002.
- [7] 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, pages 146–160. Springer, Berlin, Heidelberg, 2011. ISBN 978-3-642-19588-4 978-3-642-19589-1. URL http://link.springer.com/10.1007/978-3-642-19589-1_10.
- [8] Raffaele Conforti, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Bpmn miner: Automated discovery of bpmn process models with hierarchical structure. *Information Systems*, 56:284–303, 2016.
- [9] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12): 1281–1294, November 2008. ISSN 09505849. doi: 10.1016/j.infsof.2008.02.006.
- [10] Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A Reijers, et al. *Fundamentals of business process management*, volume 1. Springer, 2013.

- [11] Nissreen El-Saber and Artur Boronat. BPMN Formalization and Verification using Maude. In *Proceedings of the 2014 Workshop on Behaviour Modelling-Foundations and Applications*, pages 1–12. ACM Press, 2014. ISBN 978-1-4503-2791-6. doi: 10.1145/2630768.2630769. URL <http://dl.acm.org/citation.cfm?doid=2630768.2630769>.
- [12] Luciano García-Bañuelos. Pattern identification and classification in the translation from bpmn to bpel. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 436–444. Springer, 2008.
- [13] Signavio GmbH. Signavio, 2016. URL <http://www.signavio.com/>.
- [14] Heerko Groefsema and Doina Bucur. A survey of formal business process verification: From soundness to variability. In *Proceedings of the Third International Symposium on Business Modeling and Software Design*, pages 198–203, 2013.
- [15] Charles Antony Richard Hoare et al. *Communicating sequential processes*, volume 178. Prentice-hall Englewood Cliffs, 1985.
- [16] Bartek Kiepuszewski, Arthur Harry Maria ter Hofstede, and Christoph J Busler. On structured workflow modelling. In *Seminal Contributions to Information Systems Engineering*, pages 241–255. Springer, 2013.
- [17] Julius Köpke, Johann Eder, and Markus Künstner. Implementing projections of abstract interorganizational business processes. Technical report, Technical report, Universität Klagenfurt-ISYS, 2014. <http://isys.uni-klu.ac.at/PDF/2014-Impl-Process-Partitioning.pdf>, 2014.
- [18] Felix Kossak, Christa Illibauer, Verena Geist, Jan Kubovy, Christine Natschläger, Thomas Ziebmayer, Theodorich Kopetzky, Bernhard Freudenthaler, and Klaus-Dieter Schewe. A Rigorous Semantics for BPMN 2.0 Process Diagrams. In *A Rigorous Semantics for BPMN 2.0 Process Diagrams*, pages 29–152. Springer, Cham, 2014. ISBN 978-3-319-09930-9, 978-3-319-09931-6.
- [19] Ralf Laue and Jan Mendling. The impact of structuredness on error probability of process models. In *International United Information Systems Conference*, pages 585–590. Springer, 2008.
- [20] Ann Lindsay, Denise Downs, and Ken Lunn. Business processes - attempts to find a definition. *Information and Software Technology*, 45(15):1015–1019, 2003.
- [21] Rong Liu and Akhil Kumar. An analysis and taxonomy of unstructured workflows. In *International Conference on Business Process Management*, pages 268–284. Springer, 2005.
- [22] Jan Mendling, Hajo A. Reijers, and Wil MP van der Aalst. Seven process modeling guidelines (7pmg). *Information and Software Technology*, 52(2):127–136, 2010.
- [23] Jan Mendling, Laura Sanchez-Gonzalez, Felix Garcia, and Marcello La Rosa. Thresholds for error probability measures of business process models. *Journal of Systems and Software*, 85(5):1188–1197, 2012.
- [24] Shoichi Morimoto. A survey of formal verification for business process modeling. In *International Conference on Computational Science*, pages 514–522. Springer, 2008.

-
- [25] Barak Naveh. jgrapht, 2016. URL <http://jgrapht.org/>.
 - [26] Alex Norton and Rik Eshuis. Specification and verification of harmonized business-process collaborations. *Information Systems Frontiers*, 12(4):457–479, 2010.
 - [27] OASIS. Business process execution language, 2016. URL <http://bpel.xml.org/>.
 - [28] OMG. Business Process Model and Notation (BPMN V 2.0). Technical report, OMG, 2011.
 - [29] OMG. Business Process Model and Notation (BPMN V 2.02). Technical report, OMG, 2013.
 - [30] OMG. Business process model and notation, 2016. URL <http://www.bpmn.org/>.
 - [31] OMG. Unified modeling language, 2016. URL <http://www.uml.org/>.
 - [32] Oracle. Java se development kit 8, 2016. URL <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>.
 - [33] Julia Padberg, Kathrin Hoffmann, Hartmut Ehrig, Tony Modica, Enrico Biermann, and Claudia Ermel. Maintaining consistency in layered architectures of mobile ad-hoc networks. In *International Conference on Fundamental Approaches to Software Engineering*, pages 383–397. Springer, 2007.
 - [34] Fabian Pittke, Henrik Leopold, Jan Mendling, and Gerrit Tamm. Enabling reuse of process models through the detection of similar process parts. In *International Conference on Business Process Management*, pages 586–597. Springer, 2012.
 - [35] Artem Polyvyanyy, Luciano García-Bañuelos, and Marlon Dumas. Structuring acyclic process models. In *International Conference on Business Process Management*, pages 276–293. Springer, 2010.
 - [36] Artem Polyvyanyy, Luciano García-Bañuelos, Dirk Fahland, and Mathias Weske. Maximal structuring of acyclic process models. *The Computer Journal*, page bxs126, 2012.
 - [37] Hajo Reijers and Jan Mendling. Modularity in process models: Review and effects. In *International Conference on Business Process Management*, pages 20–35. Springer, 2008.
 - [38] Mehrdad Sabetzadeh, Shiva Nejati, Sotirios Liaskos, Steve Easterbrook, and Marsha Chechik. Consistency checking of conceptual models via model merging. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 221–230. IEEE, 2007.
 - [39] Bruce Silver. *BPMN method and style: with BPMN implementer’s guide*. Cody-Cassidy Press, Aptos, Calif, 2. ed edition, 2011. ISBN 0-9823681-1-9 978-0-9823681-1-4.

- [40] Marigianna Skouradaki, Katharina Göerlach, Michael Hahn, and Frank Leymann. Application of sub-graph isomorphism to extract reoccurring structures from bpmn 2.0 process models. In *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*, pages 11–20. IEEE, 2015.
- [41] Sergey Smirnov, Matthias Weidlich, and Jan Mendling. Business process model abstraction based on synthesis from well-structured behavioral profiles. *International journal of cooperative information systems*, 21(01):55–83, 2012.
- [42] 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, February 2013. ISSN 09505849. doi: 10.1016/j.infsof.2012.08.014. URL <http://linkinghub.elsevier.com/retrieve/pii/S0950584912001814>.
- [43] Jane Webster and Richard T Watson. Analyzing the past to prepare for the future: Writing a literature review, 2002.
- [44] Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Transactions on Software Engineering*, 37(3):410–429, 2011.
- [45] Peter YH Wong and Jeremy Gibbons. A process semantics for bpmn. In *International Conference on Formal Engineering Methods*, pages 355–374. Springer, 2008.
- [46] Moe Thandar Wynn, HMW Verbeek, Wil MP van der Aalst, Arthur HM ter Hofstede, and David Edmond. Business process verification-finally a reality! *Business Process Management Journal*, 15(1):74–92, 2009.

Ringraziamenti

“Arrivi puntuale all’Università, ti fai largo tra giovani e ragazze seduti sulle scalinate, ti rigiri smarrito tra quelle austere mura che le mani degli studenti hanno istoriato di esorbitanti scritte maiuscole e di graffiti minuziosi così come i cavernicoli sentivano il bisogno di fare sulle fredde pareti delle grotte per padroneggiarne l’angosciosa estraneità minerale, familiarizzarle, rovesciarle nel proprio spazio interiore, annetterle alla fisicità del vissuto. Lettore, ti conosco troppo poco per sapere se ti muovi con sicurezza indifferente all’interno d’un’Università oppure se antichi traumi o scelte meditate fanno sì che un universo di discenti e di docenti appaia come un incubo al tuo animo sensibile e sensato.”

Italo Calvino. *Se una notte d’inverno un viaggiatore*

Dopo altri due anni mi ritrovo a concludere un percorso di studi che mi ha visto cambiare radicalmente come persona. Camerino, in questo, mi ha permesso di conoscere nuove persone che, nel bene o nel male, hanno reso questo percorso pieno di emozioni.

Nonostante ciò, sento di dover ringraziare in primis me stesso, perché ogni volta che credevo di non riuscire a superare un ostacolo, ho saputo trovare la forza per rincorrere fino in fondo gli obiettivi che mi prefissavo.

Questa forza però non sarebbe stata sufficiente se in ambito universitario e personale non avessi avuto vicino insegnati, parenti e amici in grado di supportarmi.

Su tutti la donna che amo e senza la quale non sarei l’uomo che sono. Grazie Chicca, perché mi hai insegnato che la felicità la si può trovare anche negli attimi più tenebrosi, basta solo ricordarsi di accendere la luce.

Non servono parole speciali, invece, per giustificare il grazie che dedico così, senza additivi, a Mamma, Papà, Ludo e Massimo, il fratello acquisito.

Un sentito grazie lo devo sicuramente alla Prof.ssa Re che, con grande professionalità ed umanità, mi ha seguito nella stesura della tesi, insieme al Prof. Tiezzi, e che mi ha regalato diverse opportunità di crescita professionale. Più in generale, ringrazio tutti i professori che in questi quattro anni mi hanno insegnato ad amare il mondo dell’informatica.

Più che un grazie, vorrei dedicare uno *scusa* ad Andrea e Alessandro, più che due coinquilini, due anime pie, costretti a sopportare il bipolarismo che mi ha caratterizzato negli ultimi mesi.