Università degli Studi dell'Insubria
Dipartimento di Informatica e Comunicazione

Dottorato di Ricerca in Informatica
XXI Ciclo – 2005-2008

Representing Business Processes: Conceptual Model and Design Methodology

Ph.D Thesis of
Michele Chinosi

Advisor
Prof. Alberto Trombetta

Supervisor of the Doctoral program
Prof. Gaetano Aurelio Lanzarone

dedicated to my wonderful wife Gabriella

Abstract

In this work we present our contributions to business processes modeling. Namely, we have undertaken a thorough analysis of the OMG standard BPMN, along with other related technologies like WS-BPEL and XPDL. Such analysis has pointed out several weaknesses that motivate our contributions. We propose a new conceptual model of BPMN called BPeX as a clear and principled way to represent and reason about business processes. We provide a three-phase design methodology to model business processes focusing on BPMN and we introduce the notion of business process normal form. We introduce also the concept of business process views and apply them to business processes access control. Finally, we provide an extension to BPMN with privacy policies. Relevant parts of BPMN conceptual model will be included as part of the forthcoming BPMN 2.0 standard.

Acknowledgements

I would like to start thanking my advisor prof. Alberto Trombetta for constant support and collaboration in projecting and revising my thesis. For the last three years he helped me with precious recommendations and an attentive guidance in elaborating the work till the present result.
I'm also really grateful to Vishal Saxena since he gave me the opportunity to join the BPMN 2.0 submission group and for his encaouragement to submit my own contribution to the group; and to Nathaniel Palmer and the WfMC crew for involving me in the 'Architecture & Process' 2008 conference where I got in contact with many peoples and ideas.
I wish to thank also prof. Danilo Montesi who proposed this subject as a possible interesting research topic at the very beginning of my PhD and for initial cooperation in this work; prof. Gaetano Aurelio Lanzarone, supervisor of the doctoral program, having given me the chance to work on my research topic till this moment; my PhD mates I worked "desk-by-desk" with, in particular Pietro Colombo, Federico Gobbo, Stefano Braghin, Loris Bozzato, Paolo Brivio, Paola Villa, essential both for the work and the relax moments.
Finally, I thank my family for encouraging and supporting me all the time.

Thanks,
Michele

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Standards should be discovered, not invented

Vincent Cerf

Business Process Management (BPM) has been identified as one of the most important business priorities. The Workflow Management Coalition (WfMC) defines BPM as *a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships* [59]. As such, it introduces methods, tools and techniques to support the development and the analysis of operational business processes. Again, WfMC defines business process modeling as *the time period when manual and/or automated (workflow) descriptions of a process are defined and/or modified electronically* [59]. One of the most recent proposal for a business process modeling technique is Business Process Modeling Notation (BPMN), adopted as standard by OMG[1] [62].

In this chapter we introduce the definitions of Business Process Management, Business Process Modeling and Business Process Modeling Notation. Throughout this work we chose to refer to BPMN as the most promising business process modeling standard for BP representation, sharing and analysis. We sketch the main contributions this thesis provides and a selection of related works concerning the topics discussed throughout the thesis. The last section of this introduction is dedicated to the motivations behind our work and to outline our research.

[1] Object Management Group, `http://www.omg.org`

1.1 Business Processes

A *business process* (BP) is a set of one or more linked procedures or activities executed following a predefined order which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles or relationships. A process can be entirely contained within a single organizational unit as well as it can span several different organizations [59]. Business process collaboration across enterprise boundaries is a complex task due to the lack of a unique semantics for the terminology of their BP models and to the use of various standards in BP modeling and execution. Business process management (BPM) provides governance of a business's process environment to improve agility and operational performance. It is a systematic approach to improve any organization's business processes. BPM is not a technology and it is not related to diagrams creation or systems architecture.

Business Process Modeling, instead, is defined as the time period when manual and/or automated (workflow) descriptions of a process are defined and/or modified electronically [59]. Since both Business Process Modeling and Business Process Management share the same acronym (BPM), these activities are sometimes confused with each other. Business Process Modeling is the activity of representing processes of an enterprise, so that the current ("as is") process may be analyzed and improved in future ("to be") [67]. Business Process Modeling is typically performed by business analysts and managers who are trying to improve process efficiency and quality. The term "Business Process Modeling" was coined in the 1960s in the field of systems engineering. In the 1990s companies started to substitute terms like "procedures" or "functions" with the terms "processes" and "workflows".

From the 1990s to present days workflows and processes changed very rapidly [48]. In the beginning, workflows had not tools support and they were executed from single users which had to remember all the processes execution steps. In 1993 the first work sequences appeared to aid users describe the workflows and to document the processes steps. The following change concerned the passage from one single user to the distribute work among different users. In 2002 there was a big distinction between the workflows and processes representations and the backend. Users access a user interface connected to the application logic. In 2005, the possibility to change the underlying model and technologies without users take notice of the change increased the distance between the business process modeling level and the physical level. Nowadays research efforts are oriented to simplify workflows and business process modeling so that users see only the tasks they have to perform. The business process views definition which will be detailed in the Chap. 5 is a concrete mechanism to reach this goal. Related to this topic, another big asset is the separation of responsibility among different users. Each user should have his own permission to perform a set of predefined actions on the workflow (or on the business process model, depending on the adopted technology). The business process access permission policies we introduce together with business process views mechanism (in the Sect. 5.4) can be a possible solution to achieve this outcome.

The Workflow Management Coalition (WfMC) published in 1995 the Workflow

Reference Model, which can be summarized with the diagram shown in Fig. 1.1. WfMC identified five different workflows interfaces to facilitate exchange of in-

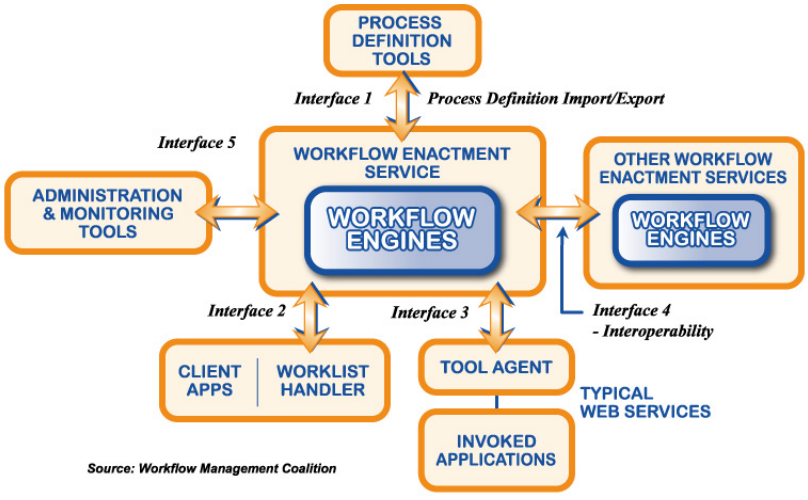


Figure 1.1: The Workflow Reference Model Diagram

formation in a standardized way, thus enabling interoperability between different products. Each interface was initially specified as a business level statement of objective, that is to say what the interface was intended to achieve in business terms and why a standardized approach was desirable. This was subsequently followed by a detailed, but abstract specification of how the interface operated and finally (for most interfaces) a "binding" specification covering the implementation of the interface in a particular technology[2], as it is possible to see in Fig. 1.2.

[2]WfMC, The Workflow Reference Model, WfMC website

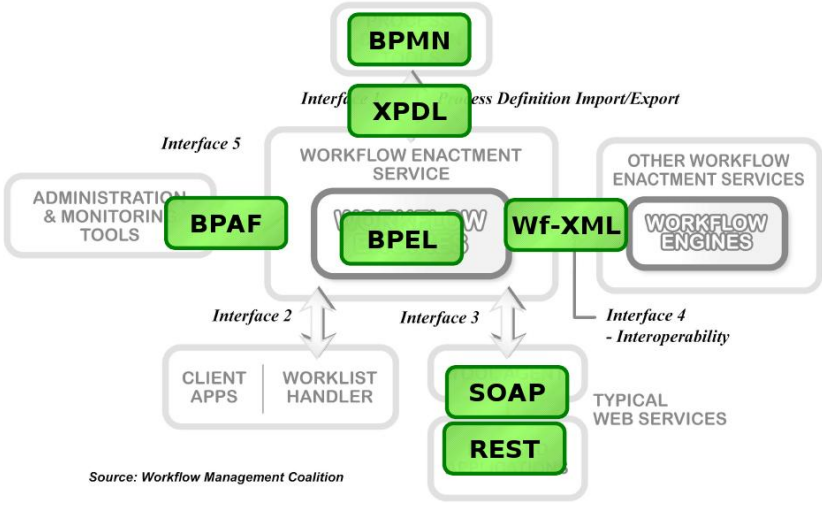


Figure 1.2: The Workflow Reference Model Diagram: Interfaces related standards

1.2 BPMN: Business Process Modeling Notation

The primary goal of BPMN is to provide a notation that is readily understandable by business users, ranging from the business analysts who sketch the initial drafts of the processes to the technical developers responsible for actually implementing them, and finally to the business staff deploying and monitoring such processes [62]. Figure 1.3 represents the present day business processes related standards scenario.

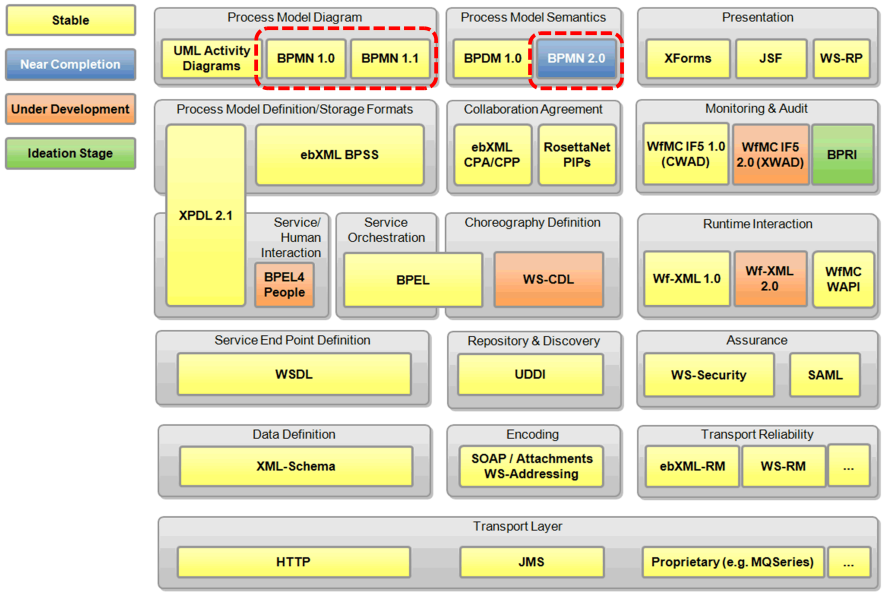


Figure 1.3: Business processes related standards overview, emphasizing the BPMN positioning

BPMN is positioned inside the Process Model Diagram subset of standards, but the forthcoming BPMN 2.0 version will span across multiple sets because of the OMG Request For Proposals requirements. BPMN was originally published in 2004 by BPMI as a graphical notation to represents the graphical layout of business processes. The ever increasing number of adoptions from companies and the growing interest upon this notation caused the adoption of BPMN as OMG standard in 2006.

BPMN provides a graphical notation in order to represent a business process as a Business Process Diagram (BPD). BPMN does not have a clearly defined semantics nor a native serialization format. BPMN 1.1 introduced an UML class diagram description of the notation to give a better formalization to the original version, but it is not enough to state that BPMN has a well-defined metamodel.

BPMN has four categories of graphical elements to build diagrams: Flow Objects, Connecting Objects, Swimlanes and Artifacts. *Flow Objects* represent all the actions which can happen inside a business process determining its behavior. They consist of Events, Activities and Gateways. *Connecting Objects* provide

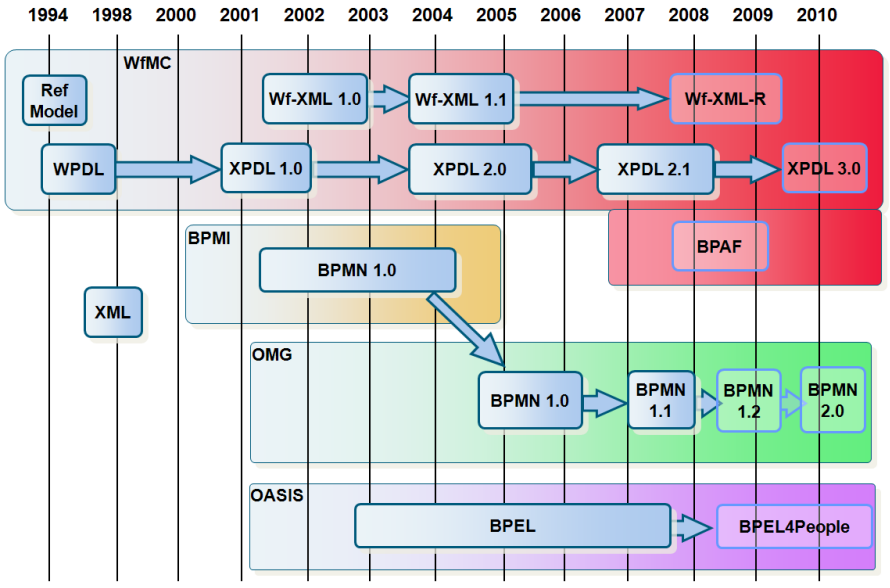


Figure 1.4: Business processes related standards timeline

three different ways of connecting various objects to each other: Sequence Flow, Message Flow and Association. *Swimlanes* give the capability of grouping the primary modeling elements. Swimlanes have two elements through which modelers can group other elements: Pools and Lanes. Finally, *Artifacts* are used to provide additional information about Process that do not affect the flow. They are: Data Object, Group and Annotation. BPMN can model three different types of business processes using three sub-models: private or internal business processes, abstract or public business processes and collaboration or global business processes. *Private business processes* generally focus on internal, organization-specific processes and are the type of processes that have been generally modeled with workflows or BPM processes (see for example [23, 54]). *Abstract processes* represent interactions between a private business process and other processes or participants, where a participant is the resource which performs the work represented by a workflow activity instance [59, 23]. An Abstract process shows only those activities involved in the interactions between two or more participants. *Collaboration processes* represent the graph of activities describing – among other things – message exchange patterns between two or more business processes. A summary of most BPMN elements is shown in Fig. 1.5[3]. Starting from BPMN 1.1 the number of the elements increases, even if most users use only a little subset of BPMN elements to model business processes, as we will analyze more in details in Sect. 4.1. For a complete description of BPMN elements and features we refer to [62].

Figures 1.6 to 1.8 show three example processes. The first one is a simple

[3]From the BPMN 1.1 poster published by Decker, Grosskopf, Wagner-Boysen, Hasso Plattner Institute, Potsdam University

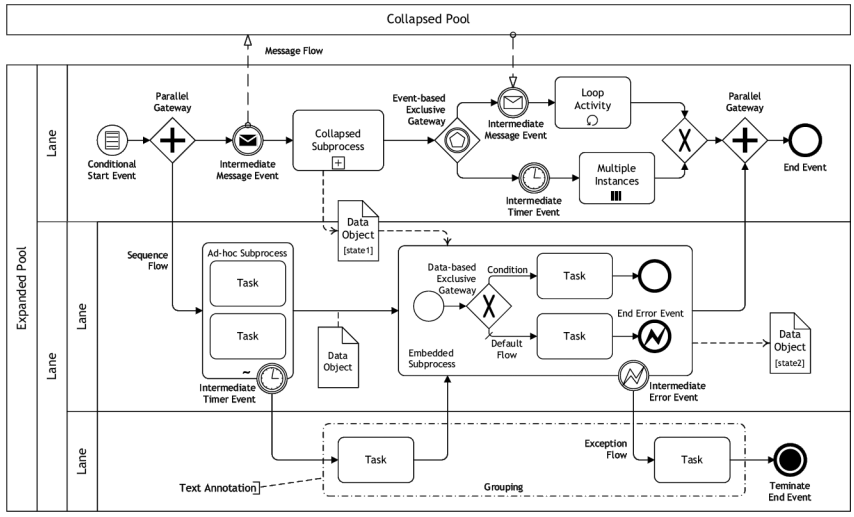


Figure 1.5: BPMN 1.1 elements summary

private process, while the second is taken from the BPMN specifications, often used as BPMN collaboration business process example. The last example is harder to read because it represents a complex business process, with plenty of elements and actors. It represents a collaboration process too, but it reveals also the private content of the Pools. As we will discuss later on in this thesis, BPMN is a powerful notation which in many cases is not used as effectively as it could be used.

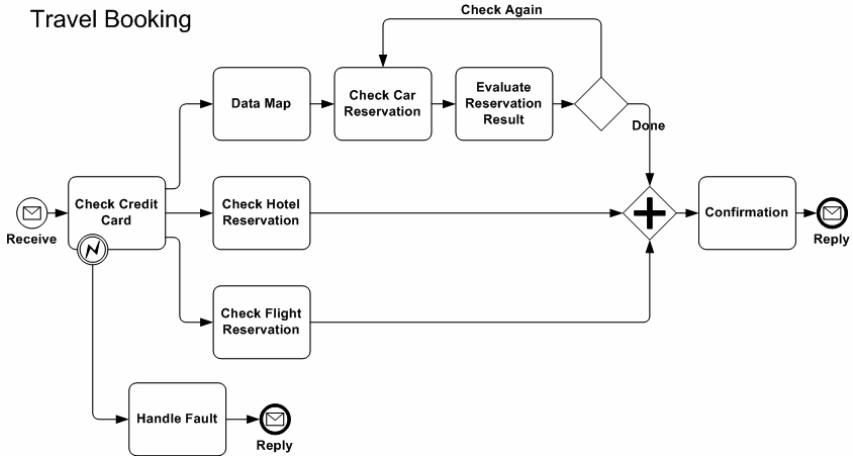


Figure 1.6: One simple *Travel booking* example modeled with BPMN

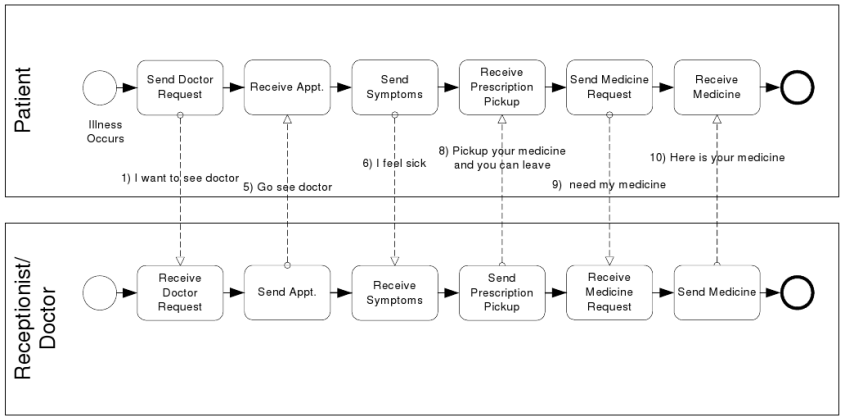


Figure 1.7: A collaboration business process depicted with BPMN

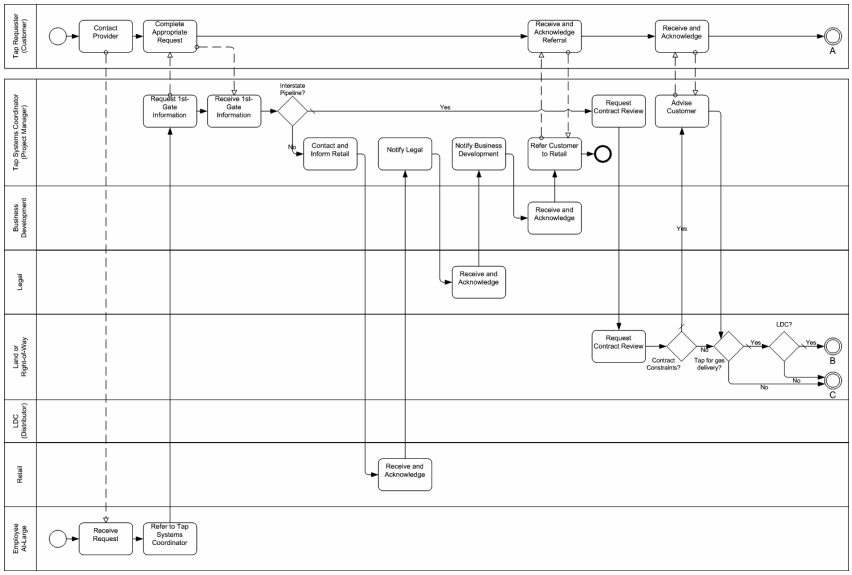


Figure 1.8: A complex collaboration business process with the Pools content shown

1.3 Our Contributions

The main question we had to deal with was: "Why should we propose a new model for BPMN?". We are aware that a novel proposal for a new BPMN core model seems not to make sense just as an OMG Request For Proposal for a new BPMN version has recently been published, with two big submission groups lobbying to prevail each other. Nevertheless, in our personal opinion, the current version of BPMN (and also the two proposals for BPMN 2.0) has some weaknesses, as discussed in full length in Chap. 3.

With this work we aim to give some concrete contributions to business process modeling. In particular we propose an alternative conceptual model for BPMN and an XML serialization of such model; we developed a business process design methodology introducing an early definition of business process normal form (BPNF) for processes which satisfy some properties; we added to business process modeling some new features like business processes views (BPDV) and user access management. In the following we briefly introduce these contributions summarizing the structure of this work.

1. **Conceptual Model for BPMN.** BPMN was originally proposed as a graphical notation to model business processes. An ever increasing number of implementations entails the need of having one more structured underlying conceptual model. There were different proposals to equip BPMN with a conceptual model, but the solution currently adopted remains unclear and too complex (BPMN 2.0 RFP calls for a single notation which defines notation, metamodel and interchange format addressing BPDM concepts). We developed our model starting from scratch, and the result of this work is a complete conceptual model for BPMN with a plain metamodel and its related XML-based serialization. We called the conceptual model BPeX and we will present it more in details in Chap. 2. Note that we refer to BPeX as model, metamodel or conceptual model in an interchangeable way depending on the context we are dealing with. We use metamodel whenever we want to address the description of the BPeX underlying model, while we refer to BPeX as conceptual model every time that some BPeX general characteristics are to be underlined. Otherwise, we generically use the term model.

2. **Comparison with Other Standards.** We provide in Chap. 3 a comparison between our conceptual model BPeX and BPEL or XPDL, as they are the most spread linearization proposals for BPMN. BPEL is a Web-services oriented business processes execution language developed by IBM and Microsoft (as BPEL4WS) and now adopted as OASIS[4] standard with the name of WS-BPEL. It is possible to refer to WS-BPEL as BPEL if it does not matter which specific version is to be considered. BPMN specifications explicitly suggest BPEL to be used as BPMN processes linearization, even if it is less expressive than BPMN. XPDL was developed by the Workflow Management Coalition (WfMC) and it has been recently updated to

[4]Organization for the Advancement of Structured Information Standards, `http://www.oasis-open.org`

fully support BPMN elements set. XPDL can store and share the graphical layout of one process but processes represented with XPDL can be executed using only XPDL compliant engines. We show how BPeX can overcome the limitations of BPEL and XPDL using a running example. The comparison concerns either the models and their XML serializations.

3. **Business Processes Design Methodology.** There exist many techniques to reengineer business processes, but a clear guidance to model from the beginning a well-formed and valid business process diagram is still missing. As such, business processes reengineering is very often a time-consuming manual activity. The starting point is to have a unique standard business process representation format, and BPMN satisfies this requirement. Now academics and industries ask even more for a methodology to model business processes. We provide in this work a business process diagrams (BPD) design methodology and the related concept of business processes normal form. This topic will be detailed in Chap. 4.

4. **Business Processes Diagrams Views.** BPMN (and also other business processes modeling languages and notations) does not provide a way to represent final users interactions with business process diagrams. Users are always considered as processes aspects to be described. In Chap. 5 we outline the concept of business process diagram views (BPDV), a new mechanism to represent BPD elements subsets which can created through the application of certain requirements and properties. We handle the problem of views updating fixing some conditions by which business process views can be updatable.

5. **Business Process Security Aspects.** As a first example of usage of views to manage final users access policies. Each view can be accessed by one or more users with different permissions, as we will outline in Sect. 5.4. We propose in Chapter 6 another example integrating BPMN (represented using our BPeX model and its XML Schema) with Web-oriented privacy policies (expressed using the standard P3P notation).

1.4 Related Works

The OMG has recently published the first version of BPMN specifications (1.1) and a Request For Proposals for the second major version of the standard (BPMN 2.0). The submission deadline was in February 2008 and there are only two submissions to the OMG RFP: the BPMN-2.0 (also known as BIOS proposal) and the BPMN-S proposals. They both include a metamodel, a graphical notation and an interchange format. The BPMN-2.0 proposal includes also a formal semantics. We will discuss more in details these two proposals in Sect. 3.5. However, during the last three years a lot of research projects have been started around BPMN. There are some interesting works regarding the new technologies developed to represent and analyze business processes. Furthermore, various tool vendors have developed their own BPMN metamodel, such as ILOG, Intalio, TIBCO, Fujitsu, Oracle, Visual Paradigm or BizAgi.

Stephen White [61] proposed a comparison on the way business process modeling notations (and in particular BPMN) express classical workflow patterns, using as correspondence the 21 workflows patterns described by van der Aalst, ter Hofstede, et al. [55]. This comparison shows that both notations can adequately model most of the patterns and that they are quite similar, aiming at the convergence of the two models. This work has been questioned by Wohed, van der Aalst et al. in [65], where the authors try to identify limitations of BPMN, to discuss how to capture the patterns in BPMN, to outline the problems identified by White's evaluation and to find out the ambiguities of BPMN specifications. They uncover the difficulties in assessing a language which does not have a formal semantics neither an execution environment. They also underline how BPMN provides support for most of the main Control Flows patterns on one hand, but on the other this support works only for nearly half of the patterns from the data perspective and it is very limited from the resource perspective. In [36, 41] ontology-based analysis of BPMN to deliver a comprehensive assessment of BPMN are presented. Starting from 2005, Dijkman, Mendling et al. published some works on semantics and analysis of BPs in BPMN [29, 27, 28, 15], where research prototypes for finding deadlocks in BPMN models are introduced. In [14, 56, 13], they proposed also a methodology to determine the similarity between BPs. These analysis confirm the relatively high maturity of BPMN even though they underline how BPMN contains some ambiguous elements. Moreover, they identify a number of critical issues related to the practice of modeling with BPMN in contemporary business process management initiatives.

There is a wide research branch which uses formal tools as for example Petri Nets, π-calculus (like in [52, 11, 66, 38]) or EPC [30, 21, 19] to provide a BPMN semantics which could be sound, but we do not follow this direction in this work.

1.5 The BPeX Roadmap

We started reasoning about BPMN properties and features in 2006, when BPMN was not yet adopted as OMG specification. In its early stage BPMN was only a graphical modeling notation to represent Business Processes. There was a lack of modeling tools and techniques for BPMN and it had a non precisely defined execution semantics.

We compared BPMN (from the current version of the specifications as well as from the two BPMN 2.0 proposals – BPMN-2.0 and BPMN-S) with all the other standard notations proposed to be used with it, either the present suggestions like BPEL, XPDL and BPDM and those ones required by the OMG RFP, such as XMI, MOF, SBVR. Our opinion is that every already existent notation was created to have particular properties and scopes. To adopt a pre-existent notation inserting it inside a new specification, although being aware that it is not a complete and correct conjunction (as in the case of BPMN and BPEL), can not be regarded as a valid solution. Now OMG calls for a unique notation which contains a conceptual model, a graphical representation and a serialization format. As we have seen from the two submission proposals, there is still an incomplete mapping between BPMN and BPEL. A native XML-Schema representation will be provided (or, at least, an XMI definition as required from the RFP), but we argue they will appear still too complex. Probably the two submission groups have to consider many other aspects, but this way the specifications become too difficult to be understood by business users.

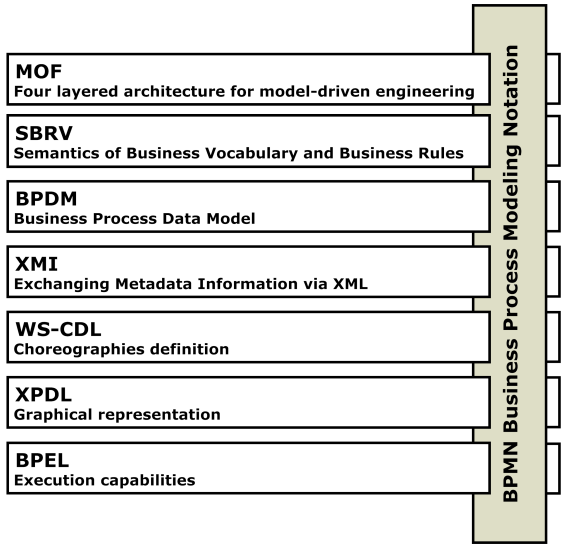


Figure 1.9: The relationships between BPMN and some other standard notations

In Fig. 1.9 we sketched out how BPMN could be related with the other standards. Each notation has a specific domain and scope as well as its own characteristics. BPMN, as defined till now, is like a transversal tool, since it aims to cover many different business process modeling related aspects such as description,

modeling, analysis and serialization. But BPMN was not conceived as a single comprehensive notation. Therefore, the necessity to connect BPMN with other emerging standards and notations, like, for example, XPDL, BPEL, XMI, BPDM, BMM, SBVR, MOF and so on comes out. However, each one of the involved notations could be used to describe some BPMN aspects, but they were published to perform also some other BPMN-independent works. At present day BPMN is spreading as a de-facto business process modeling standard and the OMG interest underlines its importance. So, BPMN needs a re-engineering process to become a complete and independent notion.

Having investigated the weak points of BPMN specifications we decided to develop our model BPeX. We do not aim to publish the next definitive BPMN specifications, but to provide instead an alternative and more simple solution which is coherent, complete and compliant, which can be used to analyze BPMN improving it with new capabilities. We decided to start building our BPeX conceptual model from scratch. We took the BPMN specifications and we traced out all the elements relationships. Figure 1.10 shows an overview of the resulting conceptual model. The BPeX conceptual model will be discussed more in details

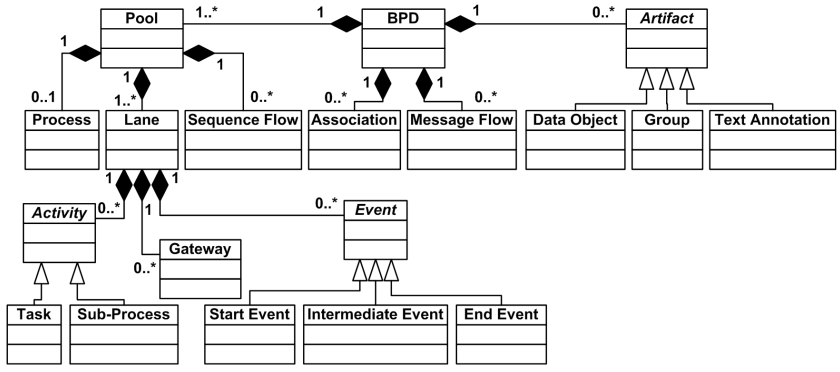


Figure 1.10: An high-level view of the BPeX conceptual model layout

in Chap. 2.

Having verified the inadequacy of other serialization formats suggested to be used to linearize BPMN diagrams (as for example BPEL and XPDL), we started developing our own serialization using XML-Schema. The first necessary step was to provide an XML representation of the BPMN attributes as they are defined in BPMN specifications. The BPMN underlying model was not suitable for representing diagrams using BPMN attributes. There were redundancies, some inconsistencies, different definitions of the same elements. So, we decided to use our BPeX conceptual model. Our XML-Schema serialization reflects the BPeX model structure.

We developed also a graphical editor for BPMN which stores the graphical diagrams using the BPeX native XML serialization, with a bijective correspondence between graphical elements and XML nodes. In Fig. 1.11 it is possible to see a screenshot of the tool. The main scope of the editor was to have a tool with the following requirements:

- syntax aiding design
- diagrams validation support
- provide a complete set of BPMN elements
- use a solid / meaningful data model
- use as much as possible standard tools and languages (SVG, XML, XML-Schema, XPath, XQuery, XPDL, BPEL)

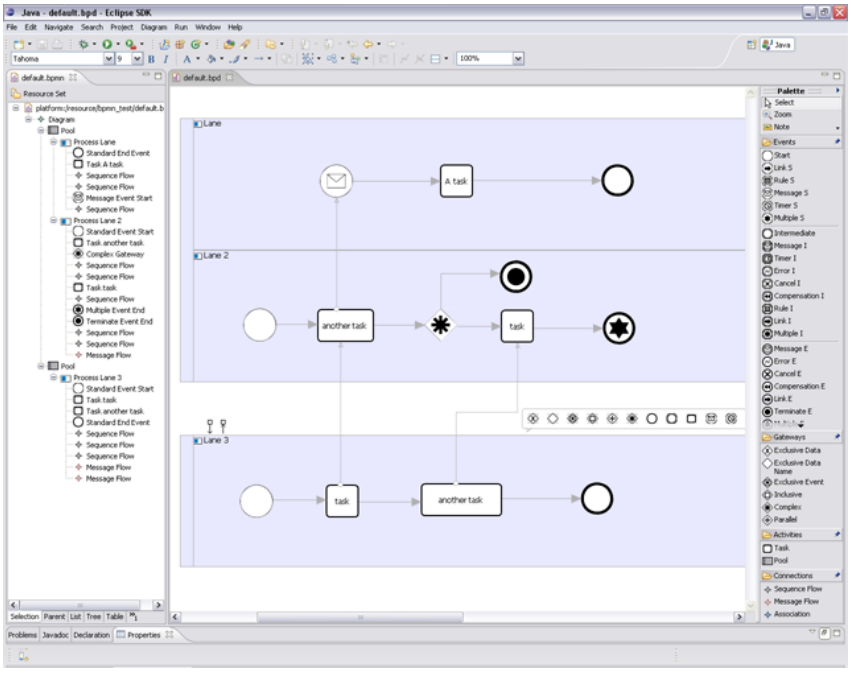


Figure 1.11: A screenshot of the BPeX editor

Figure 1.12 shows a couple of examples modeled with the editor. They are two real processes taken from a more complete and complex Italian company processes description.

Using data provided by a Business Consulting Company, we have tested our model and tool and the results are encouraging. The real-world approach in building a business process is to start from the available data and documentation and to organize them in a spreadsheet. This is typically a time-consuming manual task. For the data in our disposal, this task, applied to the two processes in Fig. 1.12 using different models, would normally require several hours. With the adoption of our model and tool, the same task required half an hour.

This underlines the very evident importance of having a graphical standard notation to describe easily-readable and -understandable processes such as BPMN. Another important aspect concerns the utility to have a graphical tool to be used

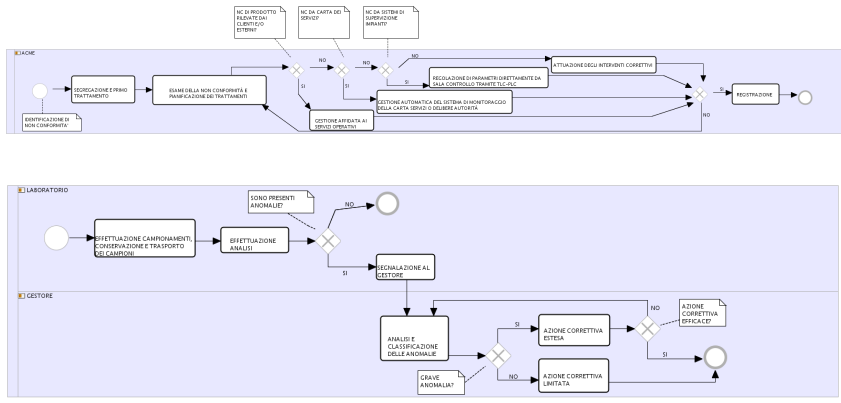


Figure 1.12: Some processes designed with BPeX editor: *Compliance check* (above) and *Anomalies treatment* (bottom)

for process modeling which could be used also to share and store the business process diagram as well as its meaning using such a model-based serialization format.

At present, academics and industries identify as the main research topics in business process management the standardization process. For the next five years they suggest as ever more important to develop methodologies [40]. So, starting reasoning on methodologies is a very relevant topic. We developed, using our BPeX model and the underlying XML serialization, a design methodology to support users defining well-formed and compliant diagrams. We will discuss in detail business process design methodology in Chap. 4.

Once we have defined and tested our BPeX model and serialization we were started looking for exemplificatory practical applications. The main business process modeling related issue not yet covered by BPMN specifications is the final user access management. Users are always considered as one of the process aspects to be modeled. There is a lack of methodology to manage final users models accesses. So we add to BPMN the capability to create diagram views and to associate to each view on one hand a set of users and on the other hand a set of permissions. Views will be defined in Chap. 5. Related to views we successfully implemented a methodology to extend business processes describing Web-related models with privacy policies definitions and their enforcement. The result of our work is a framework based on BPMN, BPeX (as a conceptual model and XML serialization) and the W3C Platform for Privacy Preferences (P3P). This application will be discussed in Chap. 6.

Chapter 2

BPeX Conceptual Model

BPMN defines a graphical notation without an explicit definition of the underlying model, made of the basic elements composing a business process and the relationships among them. Clearly, providing a BPMN model is a first, necessary step in order to precisely state what is the meaning (or, even more precisely, the behavior) of a business process described as a BPMN diagram. As such, it is an interesting problem in itself to define such a suitable, BP-oriented model [29, 27, 28]. There are some research efforts aiming at the definition of a comprehensive model representing *all* the main features of BPMN, but all such efforts were built upon already existing, but incomplete and/or inadequate formats. As we mentioned before, either BPEL and XPDL have to consider their past structures and goals. We have chosen a clean start by looking closely into BPMN and building our model from scratch, thus obtaining a clear conceptual model (or simply model, thereon), not based on previous proposals (BPEL and XPDL) and intrinsically supporting all the relevant features of BPMN. We aim to propose a complete, compact and easy-to-navigate model for BPMN, which could be adopted as the nearest serialization format to store, exchange and execute BPDs, providing also the needed mechanisms to export the model towards XPDL, BPEL and other formats.

The result of our efforts is called *BPeX* and it is defined in a top-down fashion. We start pointing out all the different BPMN symbol families, then we proceed refining them through the definition of more precise symbol families, connected in a suitably defined hierarchy. Afterward, we add a representation of flows, adopting the same top-down methodology.

We provide an XML version of such a model, in order to obtain a complete schema representing all the BPMN elements. The chosen hierarchical structure among BPMN elements (and flows) can be represented in a very natural way using XML-Schema. With a slight abuse of terminology we call BPeX both the model and its XML-based version.

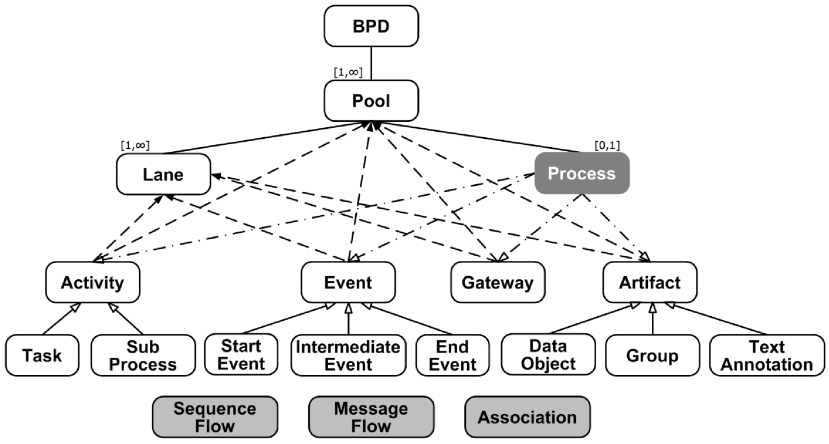


Figure 2.1: The high-level graphical representation of the BPMN model

2.1 Analysis of BPMN Model

We represent the data model, its elements and the relationships among them using the diagram shown in Fig. 2.1, which depicts the model from a high-level point of view (an UML high-level view of the BPMN metamodel can be seen in Fig. 2.3). We follow a top-down fashion methodology as the one provided by BPMN specification. Going through the definitions of the elements and their attributes, first we identify the fundamental structural entities of the model, grouped by their categories as follows: Flow Objects (Activities, Events and Gateways), Connecting Objects (SequenceFlows, MessageFlows and Associations), Swimlanes (Pools and Lanes) and Artifacts (DataObjects, Groups, Annotations). Inside Fig. 2.1 they are depicted as boxes with round corners.

For every entity, we analyze the possible specializations, which could be defined as child classes inheriting a set of properties and attributes. These entities are shown in Fig. 2.1 using boxes with round corner connected to their parent through a solid arrow with a white head (e.g., *Start*, *Intermediate* and *EndEvents* are specializations of the *Event* element).

Apart from the objects that have a graphical correspondence, BPMN provides also some other concepts, such as *BPD* and *Process*. The former describes the general properties a BPD should have. It is the root of all the other objects. The latter (depicted in Fig. 2.1 with a gray background and no boundaries to distinguish it from elements having a graphical representation) brings the processes information – such as `ProcessType`, `Status`, `Properties` – and could be referenced from *Pools* or *Independent SubProcesses*. It owns a list of all the objects that are contained within the Business Process, depicted as dashed arrows with empty heads and starting from the *Process* block.

Then we investigate the BPMN *Connecting Objects*. They represent the three connection types between the diagram entities (*SequenceFlows* for inbound paths, *MessageFlows* to connect objects of different *Pools*, and *Associations* to link *Artifacts* with other objects). There are some connecting rules that can affect the

processes behavior, the actions that could be performed over an element or a process, and the interactions between objects. *Connecting Objects* are not defined as part of any other BPMN object. Thus, we place them near the other object but without any relationships. In Fig. 2.1 they are shown with a gray background and black boundaries. Looking at the model, we focus on connections and relationships between the blocks. We gather them in three categories:

- **Strong connections**. They are the native hierarchical relationships between objects (e.g., between *Pools* and *Lanes*: BPMN specification states that a *Pool* must own at least one Lane). We depict this kind of connections with a solid line between the objects.

- **Weak connections**. These occur when two entities are linked only through attributes values (e.g., one *Activity* belongs to its *Pool* if the *Process* pertaining that *Pool* has the *Activity ID* value in its list and if the *Activity* `Pool` attribute matches the *Pool ID* value). These links are represented with dashed arrows with black heads.

- **No connections**. The *Connecting Objects* are defined outside the other objects and have a global scope over the BPDs.

As results from the analysis above, BPMN offers a weak hierarchical structure since it does not fulfill the strong hierarchical structure of the diagrams and the relationship constraints. As a matter of facts, inside a business process diagram Activities, Events and Gateways are strictly positioned inside Pools and Lanes, but the BPMN 1.1 metamodel defines Gateways, Activities and Events as children of the Flow Object class as well as Swimlanes. The same differences between graphical and model positioning of elements subsist for quite all the elements, making analysis hard to be implemented. For instance, it is not possible to design an *Activity* out of a Lane boundary. To verify this condition we need to look at the *Process* which has a reference to the *Activity* and check that all the cross-references between *Pool*, *Lane*, *Process* and *Activity* are correct. Again, to ensure that a *Sequence Flow* does not connect elements belonging to different *Pools* we need to check if either its `From` and `To` attributes values pertain to the same *Pool*. This implies also a weak referential integrity because it is not possible to enforce statically the correctness of *Sequence* and *MessageFlows* connections. Finally, as we already mentioned in Chap. 1, BPMN lacks a native interchange format that is clear, complete and adherent to the standard. Proved that neither XPDL nor BPEL can be fully adopted, we start to define from scratch a new model.

2.2 The BPeX Conceptual Model

To develop the new model we follow a top-down fashion methodology as the one provided by BPMN specifications, which starts from the root of a diagram definition (the *BPD* element) going through the definitions of the elements and their attributes. BPMN 1.1 specifications provide some UML diagrams to better specify the relationships between elements and types. There are two main aspects to be noticed. Firstly, there is no difference between elements and types: they are all defined as elements (and represented as classes in UML diagrams). Secondly, all the BPMN elements (except for *BPD* which has no connections at all) are connected through UML generalizations, this meaning that all the elements except for the root are extensions of other elements. But there are some orthogonal implicit connections which can be defined as *Connections-by-Types*. Some elements attributes are defined as having as type another element (e.g., all the *Properties* attributes are of type *Property*, which is an element child of the *Supporting Element* element). These connections could be intended as foreign keys relationships.

We aim to minimize these connections and avoid all the missing connections (as in case of *BPD*). We analyze all the elements dependencies from a hierarchical point of view as well as their graphical placements constraints. The model we define should be compliant to BPMN specification. The root of the model is the *BPD* element. Every *BPD* must have at least one *Pool* which must contain one or more *Lanes*. *Activities*, *Events* and *Gateways* must be placed inside Lane boundaries. The *Artifacts* can be positioned everywhere inside a BPD since they can be connected to every other object. Thus they can be viewed as BPD children. Quite all the elements defined by BPMN specifications as *Supporting Elements* in BPeX are treated as types rather than elements, simplifying the model structure. The connections between elements can be defined as *Connections-by-Semantics*, because we build a BPeX model considering the behavior and the possible placements of the elements inside a diagram. This structure can be easily represented with a diagram as the one depicted in Fig. 2.2. We use UML to better show differences between the BPMN original model and our proposal[1]. We use UML compositions to represent the *Connections-by-Semantics* relationships, because they adequately reflect the hierarchical positioning model of a BPD. We maintain instead the use of UML generalizations only in cases of elements extensions (like, e.g., *Events* or *Artifacts*).

Looking at the *Connecting Flow* objects, we try to find them an appropriate placement inside the model. A *Sequence Flow* can connect objects inside the same *Pool*, crossing the *Lanes* boundaries, but it must not cross the *Pool* boundaries. So it is reasonable to put *SequenceFlows* as children of a *Pool* element. Thus, it becomes impossible to define a *Sequence Flow* outside a *Pool* object. *MessageFlows* must connect objects belonging to different *Pools*. We place them as children of a *BPD* element. Finally, *Associations* link *Artifacts* to every object in the *BPD* and, thus, we put them at the same level of *Artifacts*, as children of a *BPD*.

One of the most immediate advantages of using a full hierarchical model is

[1] We use UML for comparison purposes only. BPeX is independent from any model definition format.

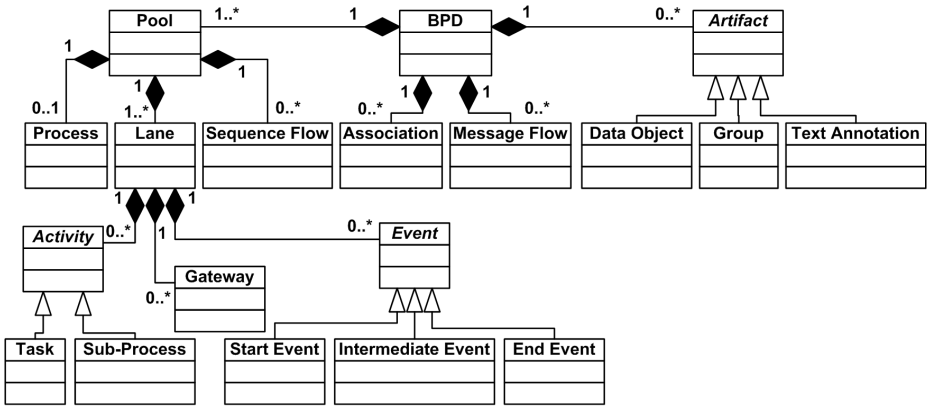


Figure 2.2: An high-level view of the BPeX conceptual model layout

the greater ease on executing queries over the process model. For instance, the complexity of checking what *Lane* an element (an *Event*, an *Activity* or a *Gateway*) belongs to depends directly on the process structure representation. Using the BPMN model provided within the specifications, a user has to control the values of the `Pool` and the `Lanes` attributes and search for the *ID* of the element inside the `GraphicalElements` attribute of the *Process* element. As it is possible to notice in List. 2.1, using XPDL the query is very complex and difficult to understand.

```
for $x in {//Activity[@Id=10]},                                          1
    $y in {//Pool[@Process = ↩                                           2
      //$x/ancestor::WorkflowProcess[1]/@Id]//Lane/@Name}                3
    return $y                                                            4
                                                                         5
Result: /Package[1]/Pools[1]/Pool[2]/Lanes[1]/Lane[1]/@Name - Lane-0      6
```

Listing 2.1: The XPath query needed to find a Lane starting from one of its contained elements using XPDL

The same listing will be presented again in Chap. 3. The BPeX definition comprehends also an XML-based serialization (to be presented in Sect. 2.3) and, thus, it is possible to use XPath/XQuery as query language. As it is possible to see in List. 2.2, using BPeX a user has just to check whether the element node is a child of the *Lane* node. This new model enhances the constraints between the objects and simplifies the main steps for a process analysis. Moreover, we clean the model from weak and confusing connections.

```
//Lane[//Task/@Id = 10]/@Name                                            1
                                                                         2
Result: /BPD[1]/Pool[2]/Lane[1]/@Name - Lane-0                           3
```

Listing 2.2: The XPath query needed to find a Lane starting from one of its contained elements using BPeX

2.3 The BPeX Linearization in XML

None of BPMN specifications (neither the 1.0 and the 1.1 version) provides any kind of direct mapping between Business Process Diagrams and a complete and native XML representation of the model. Instead, a partial mapping between BPMN and BPEL4WS is provided.

As we will discuss in Chap. 3, having a partial mapping of BPMN in BPEL is not enough. However, the Workflow Management Coalition has recently updated its XML Process Definition Language (XPDL) to fully support BPDs linearization. XPDL is the first complete BPMN mapping, even if it is not fully compliant. XPDL represents the process 'look', that is the graphical layout of a diagram. There exist also some other mappings, such as between BPMN and Petri Nets, EPC, Workflows. The advantages of having a single, coherent, complete XML representation of BPMN are very relevant since it is possible to perform in a systematic way operations like validation, analysis, refactoring, simplifying, extensions, and so forth.

As a matter of fact OMG, in its last Request for Proposal for the BPMN 2.0 specifications, requires the new version of BPMN to enable the exchange of models using XMI. The use of XML is acceptable when supported by solid motivations, because of its rapid evolution. Thus, it is possible to fit BPMN with an XML linearization.

Although these efforts, we propose here our XML serialization of BPMN 1.1 specifications. As we discussed earlier in this chapter, we need a different and simpler conceptual model for BPMN to be able to test some assumptions. Moreover, looking at the current versions of the different attempts to represent BPMN, we discovered a lot of non-suitable issues and some missing functionalities. We will deal with these comparisons in Chap. 3.

We present in this section the whole XML-Schema model, we called BPeX. The whole code is available inside the Appendix A. Firstly we sketch the structure of the XML-Schema and its relationships with our conceptual model; secondly, we analyze in details all the issues and choices we had to make to obtain a self-validating model.

The XML-Schema language applies in a very natural way for the serialization of strictly hierarchical data models, providing also a good types support to define attributes as close as possible to their original BPMN definitions. Furthermore, the use of XML-Schema eases the integration between BPeX and the other XML-based standards and the extension with new features.

We refer to the last official XML-Schema 1.0 W3C Recommendation, even though for some advanced features (like `xs:assert` and `xs:alternative`) we used the W3C Working Draft for XML-Schema 1.1 [44]. Since these features are not yet published as Recommendation they are not supported by the most spread tools providing an XML parser or an XML validator. The BPeX XML code we publish in Appendix A does not have these functions and, thus, it can be used and validated.

2.3.1 The XML-Schema Structure

The BPeX XML Schema reflects the BPeX metamodel structure. There are some differences from the original BPMN attributes definitions because of the issues we outlined in the previous section. To serialize BPMN 1.1 as it is described in the specifications means to deal with redundancies and missing information. Again, some elements (like `Categories`) are defined as elements and attribute types at the same time. We aim with BPeX serialization to provide a plain XML Schema to be used to store and exchange BPMN diagrams. It could be compared to the XPDL saving format but as it was built from scratch for BPMN and enriched to be able to represent all the BPMN features. In this subsection we discuss in details all the differences between BPMN specifications and our BPeX XML Schema structure.

Differences with the BPMN specifications

The BPMN specifications in some cases give no precise definition of the elements. This is especially due to the weak relationships between elements inside the metamodel. BPMN specifications split all the elements in two sets: the graphical elements and the supporting elements. They are all represented with UML classes, and, thus, they are all treated as elements that could be instantiated or called by other elements. The hierarchical relationships are granted only by some ID references. For example, all the elements belonging to a given Pool are identified through the `GraphicalElements` attribute of the Pool related `Process`. Every element could reference hypothetically all the other graphical or supporting elements. Therefore, it is very difficult to give an absolute position to certain elements, in particular to the abstract elements like the `Process` element. We aim not to declare the Supporting Elements set of elements, embedding them inside the graphical elements definitions.

In Fig. 2.3 it is possible to see a simplified version of the metamodel provided with the BPMN 1.1 specifications. A complete model can be viewed in [63]. In the following we analyze all the BPMN elements families one at a time, discussing the weak points we found in BPMN and providing solutions using the BPeX model instead.

Attributes default values. In many cases BPMN specifications require an attribute to be declared as mandatory, and very often it has also a default value. In XML-Schema it is not possible to declare an attribute with `use=''required''` and at the same time with a `default` statement: if a `default` clause is used, than the use of the attribute must be *optional*. However, the attribute definition in XML-Schema asserts that if an attribute has the `default` clause defined, then, if the attribute is instantiated without any value or if it is not present at all, the attribute takes the default value. Thus, everytime inside BPMN specifications an attribute is defined to be mandatory with a default value, in BPeX XML-Schema definition it is settled with `use=''optional''` together with the default value declared.

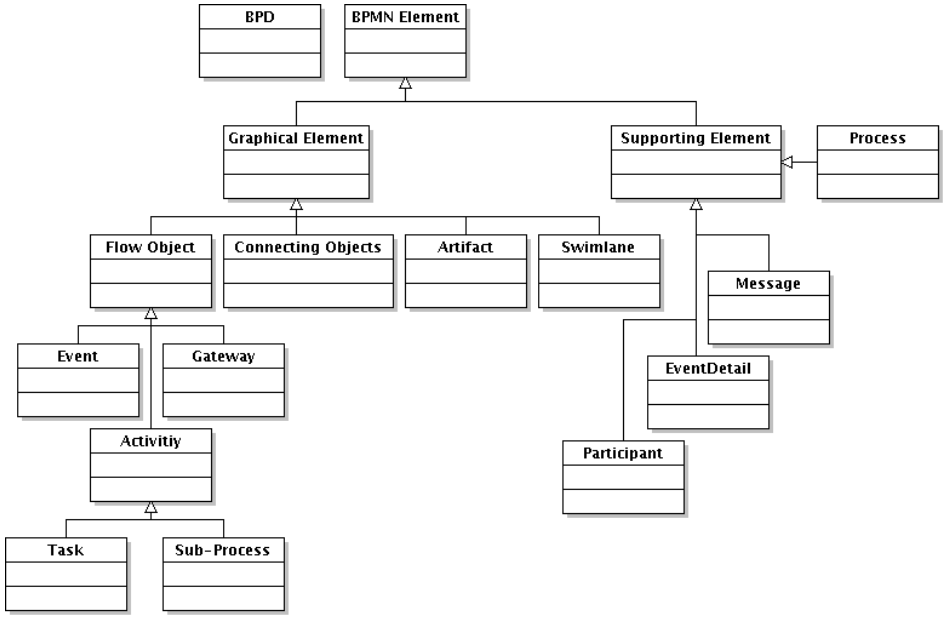
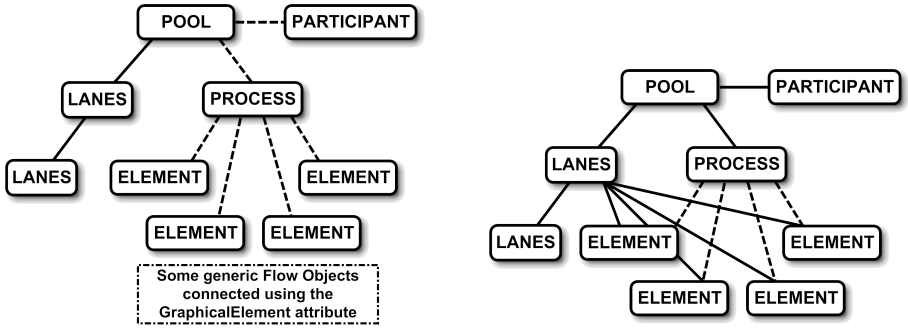


Figure 2.3: A simplified and incomplete sketch of the metamodel of BPMN 1.1

Process. The biggest difference we had to deal with is concerning the `Process` element positioning. In BPMN 1.1 UML metamodel, Processes are conceptual siblings of the BPD element, without links to/from other classes – in fact, `Processes` are defined by extension from the `SupportingElement` class, but they are conceptually independent from any other element. One `Pool` may have a related `Process` (white Pool) or not (black Pool). If it shares a `Process`, then the `Pool` element refers to it with an ID reference. The `Pool` element in BPMN 1.1 does not contain other elements but `Lanes`. The elements list (i.e., Flow Objects and Artifacts) is kept by the referenced `Process` element through a set of ID references using the `GraphicalElements` attribute. We cannot leave the `Process` element as a `BPD` element sibling, because we could have an XML valid instance with `Process` defined as root element. One Process cannot exist without a related Pool. It is the Pool element which could not have a Process, but there should not be Processes inside a BPD without related Pools. Looking at the strict relationship between the `Pool` element and the `Process` element, we decided to put the latter as child of the former. Moreover, in BPeX model we arrange all the elements inside a strong hierarchical structure to enforce the model avoiding most of the ID/IDRef connection problems and simplifying the XPath expressions. This means that, without adding other unnecessary attributes to the `Pool` element, we can find all the objects held by a `Pool`. In BPMN 1.1 specifications the `Process` element accomplishes this job. We opted for leaving this feature to the `Process` element too, thus enforcing the relationships. Figure 2.4 shows, on the left side, the BPMN 1.1 model (Fig. 2.4(a)) and, on the right side, the BPeX solution (Fig. 2.4(b)).



(a) The BPMN elements connection

(b) The BPeX proposed implementation

Figure 2.4: A comparison between the BPMN and BPeX proposals

Participant. The `Participant` element affects `Pool`, `MessageFlow` and `Web-Service` elements. It also refers to the definitions of `Role` and `Entity`. The BPMN specifications define the `Participant` as follows: *"A Pool represents a Participant"*. So, inside BPeX, we have positioned the `Participant` element as a `Pool` child. The `Participant` definition is referenced by `Message Flows`: one `Message Flow` connects two different objects belonging to two different `Pools` but, while `Message Flows` attributes require source and target objects IDs to be specified, the carried `Message` – see the next paragraph for more details – requires a source and a target `Participant` to be declared. This also easily guarantees that the two connected objects belong to different `Pools`. Likewise, one `Web-Service` must be connected to one `Participant`, also because everytime a `Message Flow` is required, an `Implementation`, having `Web-Service` as default value, has to be chosen.

Message. The `Message` definition affects `Events, Tasks` and `MessageFlows`. *"A Message is the object that is transmitted through a MessageFlow"*. So, every time there is the need of a `Message`, it is brought with a `MessageFlow`. Sometimes `MessageFlows` are not required to be graphically displayed (as in the case of `Tasks`), but they are implicit – however, this possibility is not supported in BPMN 1.1 with the definition of an attribute to specify if the `Message Flow` has to be displayed or not, unlike in the case of `Pools` boundaries. We chose to define the `Message` element inside the `Message Flow` instead of using a reference, because a `Message` can not exist without its `MessageFlow`, but it could happen to deal with one `Message Flow` without a `Message` in it.

Web Service. Web Service is required from many other elements: Start, Intermediate and End Events of type Message and Receive, Send, Service and User Tasks. Everytime messages are exchanged, an Implementation is due. The default implementation is Web Service. BPMN 1.1 defines the Web Service element as a supporting element. The only hint to be able to define Web Service element outside the Supporting Elements set is given by the example mapping between BPMN and BPEL4WS. Here the Web Service attributes are defined inside the

parent element. In BPeX we chose to instantiate a Web Service everytime it is needed. So, for example, the definition of a `Task` of type `Send` is the one in List. 2.3. To ensure that no Web Service attribute could be instantiate if the `Implementation` attribute differs from the value *Web Service*, we use an `xs:assert` statement.

```
<xs:complexType name="SendTaskType">
 <xs:complexContent>
  <xs:extension base="bpex:TaskType">
   <xs:sequence>
    <xs:element name="WebService" type="bpex:WebServiceType" minOccurs="0">
     <xs:assert test="@Implementation='Web Service'"/>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="MessageRef" type="bpex:ObjectRef" use="required"/>
   <xs:attribute name="Implementation" default="Web Service">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="Web Service"/>
      <xs:enumeration value="Other"/>
      <xs:enumeration value="Unspecified"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 2.3: An excerpt of the Send Task definition

Events. For the Events description, we aim to avoid an excessive number of definitions. In particular, the most spread linearizations of BPMN define for each Event type a different element. They provide, for example, the *Start Message Event* element, which has a different definition from the *Start Signal Event* one. Theoretically, this mechanism could be successfully implemented even though it gives a more complex XML tree. But the BPMN metamodel provides unique Event element which could belong to three different families (Start, Intermediate, End). `Message`, `Signal`, `Timer` and all the other types apply to the `Trigger` (or `Result`) attribute. They are specifications and not different elements. So, we chose to define the Events as close as possible to the original metamodel. Figure 2.5 represents the result we aim to achieve.

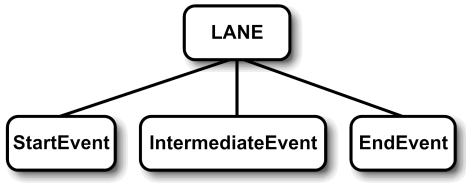


Figure 2.5: What we would like to see for the Events placement

Using the base structure of the XML-Schema file, the most suitable solution results the one depicted in Fig. 2.6, where only one Event element is defined.

It is an abstract element and, thus, it can not be directly instantiated. `Start`, `Intermediate` and `End` events are defined as substitution groups of the `Event` abstract element. But it is not possible in XML-Schema to have an abstract definition followed by substitution groups inside a Complex Type declaration (in this case, inside the LaneType definition).

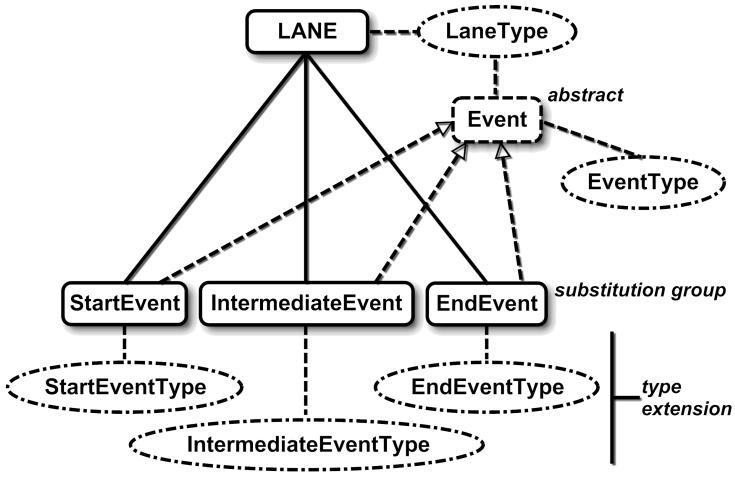


Figure 2.6: A first (incorrect) proposal for a solution for the Event placement

So, we had to find an XML-Schema compliant solution. The one presented in Fig. 2.7 is syntactically correct and adjusts the problem of the substitution groups positioning, even if it introduces an unwanted side effect. Notice that in this case the `Event` abstract definition and the `Start`, `Intermediate`, `End` specifications are placed as children of the root element. They are referenced with an `<element ref=.../>` node inside the `Lane` statement. The side effect consists of the possibility for an XML instance document to declare one of the three types of Events as a root node (e.g., the `End Event` could be the root element of the XML file describing the whole Process).

Finally, the solution we adopted is to define `Start`, `Intermediate` and `End` Events as children of the `Lane` element. Using the XML-Schema function `xs:assert` we can guarantee that the event definition matches the `Start`, `Intermediate` and `End` declaration, while using the `xs:alternative` function we can state the type of each event based on the value of its `Trigger` or `Result` attribute, as it is possible to see in the excerpt presented in List. 2.4.

Gateways. While Events are grouped into three different sets – Start, Intermediate, End – according to the time inside the Process they are instantiated and, in a second time, they are specified into one of the provided types, Gateways can be only detailed in one of the five provided types. Thus, the Gateway element is positioned as a `Lane` child. Using the `xs:alternative` function we can define only one Gateway element and then, in a second time, we can decide its type according to the `GatewayType` attribute, ensuring that the attributes defined match

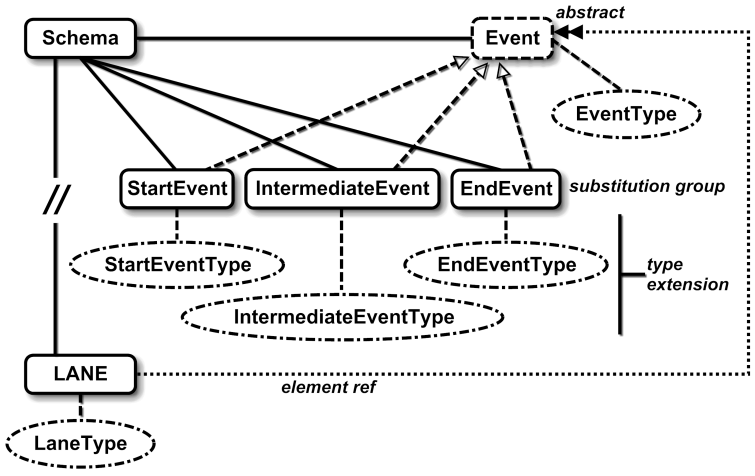


Figure 2.7: Another proposal for a solution for the Event placement

the chosen type (see List. 2.5). We had to move the default definition one level upper. The original BPMN 1.1 specifications define the default Gateway type in two steps. First, the `Exclusive` Gateway is provided as a default GatewayType. Second, if the Gateway is of type `Exclusive`, the `Data-Based Exclusive` Gateway is chosen as a default value. We have moved the `Data-Based` versus `Event-Based` choice at the same level of the other types decision. In BPeX definition there is only one default value which is the `Data-Based Exclusive` Gateway.

Artifacts. The Artifacts introduce a little problem on establishing where they have to be defined. One Artifact could be used across the whole BPD (e.g., a Group could collect elements from different Pools). The simplest solution is to define the Artifact elements set as a BPD child. Moreover, they need only one place to be instantiated. The connections with their references elements are done using the `Association` connection (for the `Data Object` and `Text Annotation` artifacts) or the `GraphicalElements` attribute (as in the case of `Group`). Thus, once we have defined an Artifact somewhere inside the XML tree, this could be used everywhere. This is the solution adopted for BPeX. There is however a toughness when a process has to deal with Reusable Sub-Processes. In this case the Sub-Process recall a Process from another BPD. In this case it is not guaranteed that also all the Artifacts will be displayed. It is possible to perform this inclusion using complicated XPath expressions, but it is not implemented in this version of BPeX. `Text Annotations` can be used by every displayed element: they are used only to better specify the operation being performed and its context or simply to provide a textual description. `Groups` are used to bundle some objects together belonging to some properties. `DataObjects`, unlike the other two Artifacts, have a role inside the Process, even though it does not affect the process flow. Objects like `Process` and `Activities` have references to `DataObjects`. For example, `Activities` have the attributes `InputSets` and `OutputSets` to specify which `DataObjects` have to be used for the Activity execution. While the `Process` ele-

```
<xs:complexType name="LaneType">
 <xs:complexContent>
  <xs:extension base="bpex:CommonSwimlaneType">
   <xs:sequence>
    <xs:element name="StartEvent" type="bpex:StartEventType" ↩
                minOccurs="0" maxOccurs="unbounded">
     <xs:assert test="@EventType='Start'"/>
    </xs:element>
    <xs:element name="IntermediateEvent" ↩
                type="bpex:IntermediateEventType" ↩
                minOccurs="0" maxOccurs="unbounded">
     <xs:assert test="@EventType='Intermediate'"/>
    </xs:element>
    <xs:element name="EndEvent" type="bpex:EndEventType" ↩
                minOccurs="0" maxOccurs="unbounded">
     <xs:assert test="@EventType='End'"/>
    </xs:element>
...
<xs:complexType name="StartEventType">
 <xs:complexContent>
  <xs:extension base="bpex:EventType">
   <xs:sequence>
    <xs:element name="Trigger" type="bpex:CommonEventDetailType" ↩
                minOccurs="0" maxOccurs="unbounded">
     <xs:alternative test="@EventDetailType='None'" ↩
                     type="bpex:NoneEventDetailType"/>
     <xs:alternative test="@EventDetailType='Message'" ↩
                     type="bpex:MessageEventDetailType"/>
     <xs:alternative test="@EventDetailType='Timer'" ↩
                     type="bpex:TimerEventDetailType"/>
     <xs:alternative test="@EventDetailType='Conditional'" ↩
                     type="bpex:ConditionalEventDetailType"/>
     <xs:alternative test="@EventDetailType='Signal'" ↩
                     type="bpex:SignalEventDetailType"/>
    </xs:element>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Listing 2.4: An excerpt of the Events definition

```
<xs:element name="Gateway" type="bpex:CommonGatewayType" ↩
            minOccurs="0" maxOccurs="unbounded">
 <xs:alternative test="@GatewayType='Data-Based␣Exclusive'" ↩
                 type="bpex:DataBasedExclusiveGatewayType"/>
 <xs:alternative test="@GatewayType='Event-Based␣Exclusive'" ↩
                 type="bpex:EventBasedExclusiveGatewayType"/>
 <xs:alternative test="@GatewayType='Inclusive'" ↩
                 type="bpex:InclusiveGatewayType"/>
 <xs:alternative test="@GatewayType='Complex'" ↩
                 type="bpex:ComplexGatewayType"/>
 <xs:alternative test="@GatewayType='Parallel'" ↩
                 type="bpex:ParallelGatewayType"/>
</xs:element>
```

Listing 2.5: An excerpt of the Gateways definition

ment could not be linked to `DataObjects` using `Associations` because it has not a graphical representation inside a diagram, `Activities` share a double link with `DataObjects`: graphically, `DataObjects` are connected to the `Activities` using `Associations`, but there is also the hard link established with the `InputSets`

and `OutputSets` attributes. Therefore, it is possible to backtrace an `Artifact` when a `Process` is referenced from a `SubProcess`. As a matter of fact, the way commonly used in graphical editors to overcome this issue is to avoid the representation of the referenced `Process` inside a collapsed `SubProcess`. When a modeler chooses to expand the `SubProcess`, then a window with the whole BPD holding the requested `Process` is shown, with all the `Artifacts` displayed.

Sub-Processes. There are three different kinds of Sub-Processes: Embedded, Reusable and Reference. The first one contains a process dependent from the parent process (the Pool Process). It does not have all the features of a complete Process, e.g., it lacks of Pools and Lanes. It contains only Flow Objects, Connecting Objects and Artifacts. All of its Start Events are of type *None*. In BPeX we represent this kind of Sub-Process using the `GraphicalElements` attribute, which maintains all the references towards the elements belonging to the Sub-Process. The elements are instantiated as descendants of a Lane, and then linked also from the Sub-Process. Its definition is not much different from the definition of the Artifact `Group`. The Reusable Sub-Process is an Activity which re-calls another Process, defined as part of another BPD, different from the Sub-Process one. The other BPD could have more than one Process, and, thus, a Reusable Sub-Process has the `ProcessRef` and the `DiagramRef` attributes. In this case is not easy to guarantee that the values provided as references to the external BPD and Process are correct. The last type of Sub-Process, the Reference Sub-Process, is used only in case of a Sub-Process holding the same elements of another Sub-Process. To spare redundant and unnecessary definitions, the attribute `SubProcessRef` points to another already defined Sub-Process.

Transactions and Signals. Some elements (like `Transactions` and `Signals`) need to be referenced from multiple objects. In these cases every time they are requested they are instantiated. They will have different IDs and so they have to share the same name and the same properties to be identified as shared objects. BPMN 1.1 provides natively this mechanism for `Transactions`, where the attribute `TransactionId` of type *String* serves as an "identifier for the Transactions used within a diagram" instead of the `ID` attribute. We extend this mechanism also to those elements which need the same capability.

2.3.2 Referential Integrity in BPeX

In a first version of BPeX XML-Schema, to gain the capability to enforce the referential integrity of the diagrams, we used the couples of `xs:key` and `xs:keyref`. As a matter of fact, `xs:ID` and `xs:IDRef` are not enough to guarantee that all the references are correct. There is not an easy way to differentiate *ID* values to ensure that every element type has its own pattern-based *ID* family (i.e., the *ID* family of *Lanes* is different from other *ID* families). This implies that, e.g., the `Lane` attribute of a *Task* – which needs to have the *ID* of a *Lane* as value – could have as value the *ID* of one *StartEvent*. Syntactically the document is valid. However, we need a more strict control on the values a reference should have. An `xs:key` can assign a unique name (the key) to every element or path inside an XML-tree.

Similarly an `xs:keyref` connects an `xs:key` value to one specific element or path. Thus, we can assure that the `Lane` attribute of a *Task* object will have as value one of the right *Lane ID* values. We use `xs:key`/`xs:keyref` also to declare which attributes an object must have according to the value another attribute gets when it is instantiated. For instance, the *StartEvent* object changes the set of its attributes according to the value of its `Trigger` attribute. If the `Trigger` attribute value is *Message*, then the *StartEvent* should have `Message` and `Implementation` as attributes and not other attributes like `TimeDate` or `ProcessRef`. To our best knowledge, the most suitable way the other serialization formats implement this characteristic is defining an element for every set of allowed attributes, introducing more complexity, redundancy and going far from the original BPMN definition.

XML-Schema 1.1 introduces some constructs which can be used instead of the couple `xs:key`/`xs:keyref`; besides, the new functions introduced here are more powerful and easier to use than `xs:key` and `xs:keyref`. We decided to use in particular two of these constructs: `xs:assert` and `xs:alternative`. Now we introduce them briefly.

xs:assert. Assert is an element which uses XPath 2.0 expressions to test some conditions. Its simplified syntax is as follows: `<xs:assert-test="XPathExpr" />`. Declaring an assert inside an element, it is possible to write all the tests that XPath 2.0 permits. Most of the BPMN attributes conditional choices can be expressed in this way. The use of assert enables all the test upon the `ID`/`IDRef` categories correspondences. As an example, we can declare the test in List. 2.6 to be sure that the `Lane` attribute of a `Task` element refers to one of the `Lane IDs` and not, for example, to a `StartEvent ID` value.

```
<xs:assert ↩                                                    1
  test="some $id in (ancestor-or-self::Lane@Id) ↩               2
  satisfies $id=@Lane" />                                       3
```

Listing 2.6: An xs:assert example from the BPeX code

xs:alternative. Alternative is an element which permits to choose the type of an element according to the success of a test performed using an XPath 2.0 expression. In particular this function can be used to test the value an attribute could assume and, according to this value, to choose the right element type to apply. This function is very useful if used with the `Events`, `Gateways` and `Activities` elements. In fact, it is possible to declare a `StartEvent` as a child element of a `Lane`, selecting its attributes using the `xs:alternative` clause. This makes a significant difference with respect to the other purposes like XPDL or BPEL4WS, where every element type is treated as a different element. BPMN specification define Gateways, Events and Activities as elements, but not their types. For instance, a `Message Start Event` is a kind of `StartEvent` and not a different element. An `xs:alternative` example can be viewed in List. 2.7.

```
<xs:element name="Trigger" type="bpex:CommonEventDetailType" ↩
            minOccurs="0" maxOccurs="unbounded">
  <xs:alternative test="@EventDetailType='None'" ↩
                  type="bpex:NoneEventDetailType"/>
  <xs:alternative test="@EventDetailType='Message'" ↩
                  type="bpex:MessageEventDetailType"/>
  <xs:alternative test="@EventDetailType='Timer'" ↩
                  type="bpex:TimerEventDetailType"/>
  <xs:alternative test="@EventDetailType='Conditional'" ↩
                  type="bpex:ConditionalEventDetailType"/>
  <xs:alternative test="@EventDetailType='Signal'" ↩
                  type="bpex:SignalEventDetailType"/>
</xs:element>
```

Listing 2.7: An xs:alternative example from the BPeX code

Chapter 3

A Comparison with Other Standards

> All models are wrong, some are useful.
>
> *George Edward Pelham Box*

We compare in this chapter the two more adopted solutions for serializing BPMN (BPEL and XPDL) with our proposal. There are many discussions around this topic, and it is possible to easily find clearly different positions. We will focus only on BPEL and XPDL because, up to now, they are the only two widely adopted solutions to serialize BPMN diagrams using XML.

There are some works on the correlations between BPMN and BPEL [37, 64, 53, 33, 34] and XPDL [60, 46, 18], focusing on the expressive power of the two XML-based representations and on the lack of a clear meta-model. BPEL is an XML-based 'execution language' to represent Web-services orchestrations. Originally developed by Microsoft and IBM, it is now an OASIS standard. It is an 'execution language' in the sense that business processes serialized in BPEL can be executed by a software tool. XPDL, instead, is an XML-based process design format developed and published by the Workflow Management Coalition (WfMC) as a definition format for workflows and business processes. It is typically used to serialize the graphical layout of one process. The fact that BPMN and BPEL-XPDL are independent projects raises many issues about how much the latter is fit to serialize the former, as widely discussed by Bruce Silver and Keith Swenson in their blogs [45, 47]. The discussion focuses on the definition of meaning and behavior of a BPMN diagram, in particular if it is possible to state that the meaning of a BPMN diagram depends on the graphical layout – in case, XPDL should be the better choice for serializing BPMN – or on the execution information – like using BPEL.

From its very beginning, BPMN chose BPEL as its serialization format, despite its clear limitations and some different design decisions. As it is possible to see in Fig. 3.1, BPEL was launched in 2002 (as BPEL4WS), before the beginning of BPMN project. There was no effort to modify or to extend BPEL to fully support BPMN. It has been derived from the Microsoft, IBM and BEA efforts,

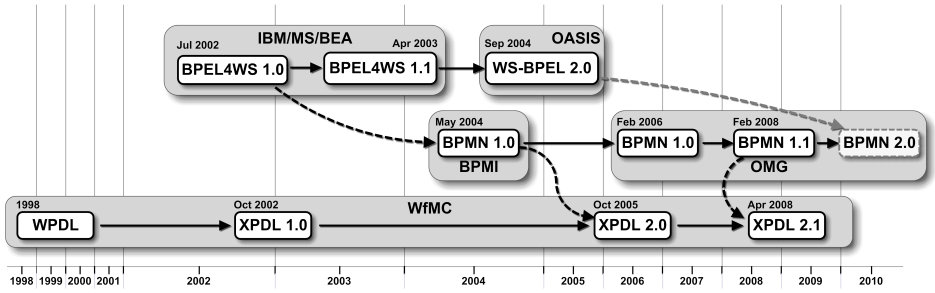


Figure 3.1: The BPMN, BPEL and XPDL time line

starting from other proposals like the Microsoft XLANG and the IBM WSFL, with a contribution from Intalio BPML language. At the present moment (late 2008), while BPMN is an OMG standard, BPEL continues to be developed by its group as an OASIS standard project. It changed its name in WS-BPEL to follow the Web-Service standard naming convention.

The Workflow Management Coalition, instead, decided to extend XPDL to fully support the BPMN notation. As it is possible to see in Fig. 3.1, XPDL remained independent from BPMN until October 2005, seven years after it was launched. From XPDL 2.0 on, its relationships with BPMN follow only one single direction from BPMN towards XPDL. So, we can see a directional influence flow passing through BPEL, BPMN, and XPDL (the dashed arrows in Fig. 3.1). BPMN influenced XPDL and was influenced by BPEL.

Summing up, we face with three different standard notations, each one with its properties and scopes.

- **BPMN** is a graphical notation for representing business processes. It is a standard to represent the 'look' of a process.
- **BPEL** is an 'execution language' for the definition of Web-services orchestration. It is independent from BPMN.
- **XPDL** linearizes the process diagrams. It was conceived as a format for storing and exchanging diagrams. It is a process design format extended to support BPMN.

To accomplish the task of comparing our BPeX proposal with both BPEL and XPDL, throughout this chapter we will consider the Electronic Order example process depicted in XPDL specification[1]. The process represents an order entry system in which an order enters the system as a formatted string. The output of the process is another string that indicates whether the order was confirmed or rejected. The process is composed of a main process called *EOrder* and two sub-processes called *CreditCheck* and *FillOrder*.

In Fig. 3.2 the *EOrder* process has been redrawn using BPMN. The diagram is quite similar to the one presented in XPDL specification with the exception of Pool and Lane boundaries which in this case were explicitly drawn. For a more

[1]XPDL 2.0 (Oct 2005), Section 8.1, pages 109-127

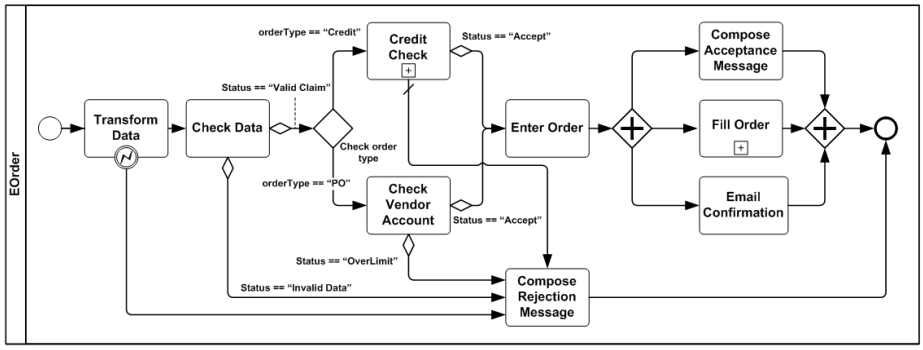


Figure 3.2: The *EOrder* example process

detailed description of the process, its steps and for the BPMN representation of its sub-processes, refer to [60].

We will conclude this chapter providing an analysis of the emerging BPMN 2.0 proposals. We are directly involved in one of the two groups *"lobbying the ascendancy"*[2] of their proposals. We will summarize the issues we submitted to the technical committee, together with a comparison between this BPMN 2.0 proposal and our BPeX model. Before to proceed with the comparison between BPMN and BPEL or XPDL we briefly introduce here the two languages.

Introduction To BPEL. WS-BPEL is an acronym for Web Services Business Process Execution Language, a revision of the original acronym BPEL4WS (Business Process Execution Language for Web Services). WS-BPEL 2.0 specification (BPEL) defines a language for business process orchestration based on web services. BPEL is an orchestration language and not a choreography language. Among other design goals, BPEL is intended to define business processes using an XML-based language but without to define a graphical representation of processes or provide any particular design methodology for processes. The reference specifications version we refer to throughout this thesis is WS-BPEL 2.0, published in April 2007.

Introduction To XPDL. The XML Process Definition Language (XPDL) is a format standardized by the Workflow Management Coalition (WfMC) to interchange Business Process definitions between different modeling tools and management suites. XPDL includes a metamodel for describing the process definition and also a companion XML-Schema for the interchange of process definitions. The WfMC has identified five functional interfaces to a process as part of its standardization program, as we already introduced in Chap. 1. This specification forms part of the documentation relating to the Interface one, supporting Process Definition Import and Export. XPDL is designed to exchange the process defini-

[2] *BPMN 2.0 - Marriage Made In Heaven or Trough of Disillusionment*, from BPMfocus website, `http://bpmfocus.wordpress.com/2008/10/31/bpmn2/`

tion, both the graphics and the semantics of a workflow business process. XPDL contains elements to hold graphical information, such as elements positioning, as well as limited executable aspects which would be used to run a process with an XPDL compliant engine. The reference specifications version we refer to throughout this work is XPDL 2.1, published in October 2008, except where otherwise specified.

3.1 BPeX Compared to WS-BPEL

First of all we serialize the example process using BPEL. As we already mentioned in the introduction of this chapter, BPEL is a BPMN-independent standard, suggested by BPMN specifications to be used for some BPDs translations, although BPEL and BPMN have two completely different purposes. In fact, the depicted example process can not be rewritten directly following the instructions provided by BPMN specifications without some tricks, and that is why BPEL does not care about the graphical representation of the diagram but it is oriented toward the definition of the execution behavior.

As we already mentioned earlier, there exist several works discussing on the differences between BPMN and BPEL under different aspects [37, 64, 53, 33, 34]. There are many BPMN elements and attributes that have no mapping to any BPEL element or attribute. For instance, *Pools*, *Lanes*, *MessageFlows*, *Independent SubProcesses*, *Artifacts*, *Associations*. Also some attributes like *Id*, *Categories* and *Documentation* do not have a correspondence in BPEL. Most of the *OR Gateways* are serialized with nested *If-[ElseIf]-Else* (the old *BPEL4WS switch*) conditional statements, while *AND (Parallel) Gateways* become BPEL *flows*, structured as sequences of elements. Naming is inconsistent between the two approaches. The correspondence is established from the BPMN elements names and the values of some BPEL activity labels, paying attention to the renaming process required from BPEL. That is, activities names and attributes can not contain spaces and special characters.

Once we have the XML serialization of the example process, we represent it graphically to navigate more easily its structure and to better infer some properties. The Fig. 3.3 is a sketch of the graphical representation of the example process model. A bigger figure can be seen in Appendix B.

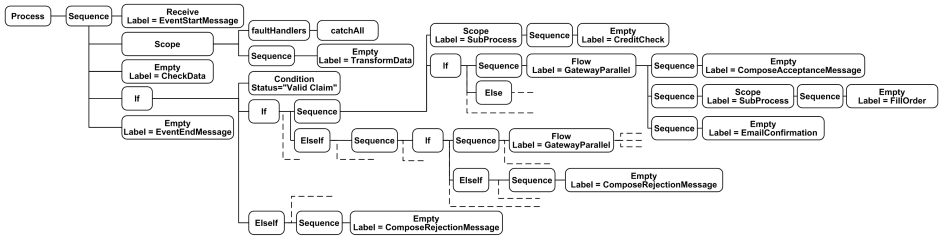


Figure 3.3: A sketch of the graphical representation of the XML-tree structure for the BPEL serialization of the running example

The key point is that BPMN does not have an execution semantics, while BPEL has. So, BPMN can not be transformed into BPEL: if the BPMN diagram contains enough information to be directly transformed into an executable BPEL process, then it contains enough information to be executed directly. The transformation does not produce information on its own.

Notice that the model of the BPEL serialization mainly focuses on the execution paths rather than on the activities flows as in BPMN description of the process. Looking at the Fig. 3.3 it is clear how articulated is the BPEL tree struc-

ture, considering that the depicted nodes are relative only to a little piece of the process, adding complexity and raising the probability to do errors. Moreover, BPEL does not contain elements to represent the graphical aspects of a process diagram. That is, serializing one BPMN process to BPEL there occurs a loss of information, both graphical and behavioral. There are effectively some BPMN diagrams that can not be translated in BPEL. As we depicted above, BPEL can not represent multiple BPMN processes using one XML tree but it needs to map each process separately to as many XML trees. BPMN specifications themselves state that: "it is possible to represent processes in BPMN that cannot be mapped to BPEL".

They are two different formats for two different purposes, merged together by BPMI to provide BPMN with an already well-known Web-Service interface. As a matter of fact, the BPMN 2.0 RFP calls for a unique model and notation definition, aiming to reach a directly executing BPMN diagrams.

An interesting question is what we call "limitations of BPEL": could there instead be limitations with the method used to transform BPMN to BPEL? But there is no definitive way to transform BPMN to BPEL. The two currently proposed BPMN 2.0 specifications clearly state that still not all BPMN processes can be mapped to BPEL.

3.2 BPeX Compared to XPDL

Starting from the XPDL linearization of the process, as it appears in the XPDL specifications, we sketch part of its model in Fig. 3.4, as we made for BPEL in the previous section: it represents only some activities from the first XPDL *WorkflowProcess* and few local transitions, that is *Sequence Flows*. Notice that the XPDL tree model is very intricate and not easy to read, but it is clear also in this case how complex and detached from the BPD is the XML serialization structure. Among other things, this is because XPDL has to maintain a backward compatibility with its previous version, which was not supposed to represent processes modeled with BPMN (XPDL raises from the old WPDL, published by WfMC in 1998). Leading off from its 2.0 version, XPDL supports the whole set of BPMN graphical elements, but with some limitations. A bigger figure can be seen in Appendix B.

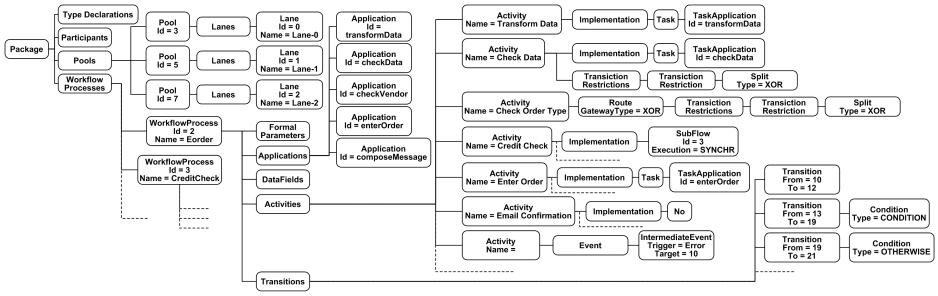


Figure 3.4: The simplified version of the graphical layout of the XPDL tree model for the running example

First, BPMN elements are tied up with the native XPDL elements. To avoid an excessive redundancy on elements names and behaviors, WfMC suggests to remap some BPMN elements with XPDL ones (e.g., the BPMN *Process* element becomes the XPDL *WorkflowProcess* element, *SequenceFlows* become *Transitions*, *SubProcesses* become *SubFlows* and *Gateways* become one of the XPDL *Route*, *Split*, *Join*, ... elements depending on their behaviors). Thus, as in BPEL, there is not a bijective mapping between BPMN and XPDL elements names and attributes. Some BPMN elements inside the XPDL model are positioned in unsuitable positions, different from their original ones (e.g., *Tasks* in BPMN are specifications of an *Activity*, while in XPDL *Tasks* are children of an *Implementation* element, descendant of an *Activity* block which belongs to a common *Activities* element). Table 3.1 sketches some of these differences.

Moreover, XPDL has a weak native referential integrity. Almost all the elements in XPDL have not a unique identifier. All the `ID` and `IDRef` attributes are defined inside the XPDL specifications and in the XML-Schema model as of `xsd:NMTOKEN` type, while the XML-Schema standard provides the `xsd:ID` and `xsd:IDREF` types. Besides, BPMN specifications require the `ID` field to be "a unique Id that identifies the object from other objects within the Diagram". This is a serious flaw: as we can see in Listing 3.1, in the XPDL linearization of the

BPMN	XPDL
Processes are children of Pools element	Pools are defined separately from WorkflowProcesses, referencing them through relationships
Tasks are specifications of Activity elements	Tasks are children of an Implementation element, descendant of an Activity block which belongs to a unique Activities element
Events are directly referenced from a Lane element	Events are children of an Activity element
One Gateway is defined within a Lane	The Route element is defined as child of an Activity

Table 3.1: A comparison between some BPMN and XPDL elements positions

example process there are one *Pool*, one *Lane* and one *WorkflowProcess* having the same value – i.e., the value **2** – for their own `ID` attributes, as it is possible to see in the code excerpt in Listing 3.1, lines 1,6 and 12 and in Fig. 3.5.

```
<Pool Process="1" Id="2" BoundaryVisible="false">                    1
 <Lanes/>                                                            2
</Pool>                                                              3
<Pool Process="4" Id="7" Name="" BoundaryVisible="true">             4
 <Lanes>                                                             5
  <Lane Id="2" Name="Lane-2" ParentLane="7">                         6
  </Lane>                                                            7
 </Lanes>                                                            8
</Pool>                                                              9
</Pools>                                                             10
<WorkflowProcesses>                                                  11
 <WorkflowProcess Id="2" Name="EORDER">                              12
  <ProcessHeader/>                                                   13
</WorkflowProcess>                                                   14
```

Listing 3.1: An excerpt of the XPDL code with ambiguous *ID* values

Furthermore, it is feasible to assign to different elements of the same type an identical value for the `ID` attribute (e.g., define three different Lanes having the same `ID=2`). The result is an XML instance which is valid with respect to the XPDL XML-Schema model[3]. Defining `IDRef` attributes as `xsd:NMTOKEN` introduces the risk to have empty or wrong references. The `xsd:IDREF` type requires the value of an `IDRef` attribute to match an existing `ID` value. Try to change an `IDRef` value of an element – part of the XPDL process example – with a non-existent one and the XML tree is still validated with respect to its XPDL XML-Schema model. For instance, we can consider the *Pool* element, having an attribute called *Process* and defined in XPDL as follows: "The Process attribute defines the Process that is contained within the Pool". Thus, it is reasonable to assume that the value of the *Process* attribute should be a reference to a *WorkflowProcess* identifier. Instead, it is possible to declare as a `Process` value a non-existent string and the document remains valid (see Fig. 3.6). This means that the diagram could have a *Pool* referencing to a non-existent *Process*. This

[3]here, *valid* is used with the W3C meaning of an XML file that is in compliance with its related XML-Schema model

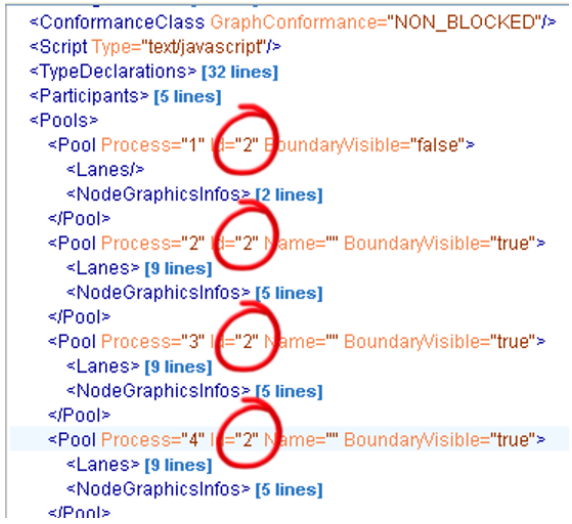


Figure 3.5: Multiple elements sharing the same ID value

flaw has not been solved, also because all the editors and tools having an XPDL-compliant engine use a parser, and, thus, this is not a priority of WfMC at this moment to enrich XPDL with referential integrity constraints.

Figure 3.6: An element with a non existent IDRef valuey

This weakness implies the necessity to write very complex queries. Listing 3.2

shows the XPath query needed to find which *Lane* a given *Task* belongs to. This is a common interrogation one modeler could perform against a BPD, and the complexity is very high if compared to the simplicity of the request. Notice also the result format, which is quite meaningless in helping the modeler to find the sought result inside the diagram (e.g., there are no *Package* elements drawn in the BPD).

```
for $x in {//Activity[@Id=10]},                                          1
    $y in {//Pool[@Process = ↩                                           2
      //$x/ancestor::WorkflowProcess[1]/@Id]//Lane/@Name}                3
    return $y                                                            4
                                                                         5
Result: /Package[1]/Pools[1]/Pool[2]/Lanes[1]/Lane[1]/@Name - Lane-0      6
```

Listing 3.2: The XPath query needed to find a Lane starting from one of its contained elements

It has been stated that these are not real problems, since nobody writes XPDL code by hand. Modelers, companies, business processes engineers always use a modeling support tool. Using a software tool relieves the modeler to think about the referential integrity constraints as well as about other conditions which have to be satisfied since the checks are done by the tool itself. Modelers and users see an easy tool aiding them during the design time, giving some suggestions or denying some operations. In this way, it does not really matter to have a sound underlying model. Rather, much of the burden is laid upon the modeling tools, which are not standardized and offer on guarantee of a correct execution when performing checks.

3.3 How BPeX Copes with the Motivating Example

As we will see, adopting the BPeX conceptual model will result in a more faithful representation of the business process components of the example under scrutiny. The underlying idea which has driven our efforts was to implement a BPMN native serialization format, which could be used to represent all the BPMN diagrams without limitations and which could be executed to give to BPMN the direct execution capability, now aimed by OMG with the BPMN 2.0 RFP. Using the BPeX model we outlined in Chap. 2, we represent the example process as follows. The whole Business Process example is entirely contained inside a BPD. It has three *Pools*: the *EOrder* represents the main process, while *CreditCheck* and *FillOrder* are referenced by two *SubProcesses*. *Pools* blocks are children of *BPD*. Every *Pool* has one single *Lane*, one *Process* and some *Activities*, *Events* and *Gateways* connected with *Sequence Flows*. All these objects are children of their respective *Pools*. Inside some blocks we write the values of certain attributes to show the direct relationship with those provided by BPMN specifications. A high-level graphical portrayal of the same subset of elements we have already shown in BPEL and XPDL sections can be viewed in Fig. 3.7.

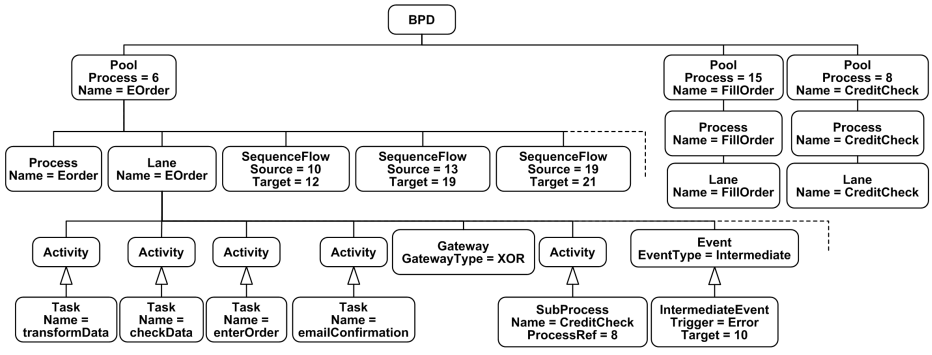


Figure 3.7: An overview of the graphical representation of the BPeX suggested solution for the running example

It is important to notice that in the BPeX model on one hand there are no other elements in addition to those required to serialize the business process diagram (unlike the XPDL proposal), and on the other hand, there is a bijective correspondence between the nodes names of BPeX model and BPD elements ones (unlike the BPEL solution).

The BPeX model diagram structure has fewer elements, improving diagram readability and thus it is more easily understandable than BPEL and XPDL ones we have shown respectively in Figg. 3.3 and 3.4. It better reflects the BPD elements placement, implicitly underlining some constraints, especially on the flows. Moreover, we can serialize all the elements of the BPD, without the need of renaming conventions or some other substitutions.

Moreover, using BPeX we can guarantee a native referential integrity, based

```
//Lane[//Task/@Id=10/@Name                                                    1
                                                                              2
Result: /BPD[1]/Pool[2]/Lane[1]/@Name - Lane-0                                3
```

Listing 3.3: The XPath query needed to find a Lane starting from one of its contained elements using the BPeX model

on its more simple and precise structure and on the use of XML-Schema types and constraints. As an example, in List. 3.3 it is possible to see the XQuery code needed to answer the query: "Which Lane does the Task with `Id=10` belong to?"

```
for $x in {//Activity[@Id=10]},                                               1
    $y in {//Pool[@Process = ↩                                                2
      //$x/ancestor::WorkflowProcess[1]/@Id]//Lane/@Name}                     3
    return $y                                                                 4
                                                                              5
Result: /Package[1]/Pools[1]/Pool[2]/Lanes[1]/Lane[1]/@Name - Lane-0           6
```

Listing 3.4: The XPath query needed to find a Lane starting from one of its contained elements using XPDL

The query is the same presented in Section 3.2, which we suggest again in List. 3.4 for convenience. Notice that using BPeX the query is very simple and the result yelds the position of the searched elements inside the diagram, steering the user and saving time. For instance, looking at the example, the result element is positioned (reading the result of the query from left rightward) inside the first (and unique) BPD, then in the second Pool and finally the result is the first Lane.

3.4 A Brief Summary of XPDL, WS-BPEL and BPeX Features

In this section we summarize pros and cons of the two modeling formats for BPMN we have investigated in respect of our BPeX model. Table 3.2 compares the models under some important aspects as we have already mentioned in the previous sections. Looking at the concepts summarized in the table, we lay out them more in details. When we talk about possibilities of combining BPeX with BPEL or XPDL or any other standard, we understand to import in the BPeX XML-Schema the XML-Schema of the other standard (or just the needed part), eventually marked with a different namespace to avoid possible elements or types redefinitions.

	BPEL	XPDL	**BPeX**
Expressive power	Less expressive	More expressive	Bijective correspondence
Naming convention	Names are different	Some names different	No differences
Structure of the model	Completely different	Some relevant differences	Few adjustments due
Native Referential Integrity	Partially	Missing	Strong
Execution capabilities	Full support	No execution allowed	Not yet but planned
Graphical information	Not at all	Full support	With extensions
Analysis	Very hard if they concerns BPMN models	Many queries required	Few simple queries
Extensions to BPMN	Hard to implement	Very difficult to extend the model	Successful experiments done

Table 3.2: A comparison between BPEL, XPDL and BPeX features

Expressive Power. Both BPEL and XPDL have not been developed to fully represent BPMN. The former was chosen by BPMI first and by OMG in the following as a serialization for BPMN although it misses most of the main objects. The latter was extended to fully support the BPMN elements set – and, thus, it is possible to state that XPDL is compliant with the BPMN graphical elements definition – but the need of a backward compatibility introduced some significant differences. BPeX was built from scratch and, thus, it is in a bijective relationship with BPMN elements definition.

Naming Convention. BPEL names BPMN elements names with value-based labels, with a loss of expressive power and forces the modeler to remap objects. Moreover, elements names must trim spaces and any special character, with the risk of having two elements sharing the same name. XPDL renames only those elements which were already provided by its earlier versions. BPeX uses instead the original names without any modification.

Structure of the Model. BPEL has its own structure and BPDs needs to be adapted in order to be serialized. XPDL puts BPMN objects inside its structure. This is closer to BPDs graphical representation but it is still far from being a direct mapping of BPDs structure. With BPeX we follow very closely BPMN specifications and, thus, we build a fully compliant model to BPDs structure.

Native Referential Integrity. BPEL has its own structure and so the controls are upon the element as they are positioned inside the tree. There are some strict controls but they pertain only partially to the BPMN structure verification. As we analyzed in Sect. 3.2, XPDL has a weak referential integrity. BPeX uses all the capabilities XML-Schema provides to define a model which can be statically validated. The referential integrity support in BPeX guarantees that BPDs which are serialized with BPeX are fully compliant with BPMN specifications.

Execution Capabilities. BPEL is the most important execution language for web services and business processes, as opposed to XPDL which can not be executed at all. BPeX was conceived to be extensible and it is a promising future work to define a suitable executable format.

Graphical Information. XPDL was conceived to be a description language for workflows. It can intrinsically bring all the graphical information that allows the rendering of a workflow (and of course a business process) with different tools and in different contexts. BPEL mainly focuses on the execution information and, thus, it does not save any graphical information. BPeX saves the hierarchical structure of a BPD and can be combined with XPDL to save also graphical information for every object.

Analysis. It is highly non-trivial work to analyze a BPD modeled with BPMN and then serialized in BPEL. If the original model is composed of more than one *Pool* we have to look for more than one BPEL process. Besides, BPEL is less expressive and needs names remapping, and this makes it very difficult to inspect. Using XPDL makes quite difficult performing queries upon elements and flows. The XPDL tree structure is very articulated and thus queries need to be very complex and expensive. The BPeX tree structure was built upon the BPMN one and it is very easy to check structural properties like element inclusion.

Extensions. All the three models are written in XML-Schema language and, thus, they have the capability to be extended and enriched with new features. However, since BPEL and XPDL have a very complex and articulated structure, especially if considered in relation to represent BPMN diagrams, BPeX appears to be a good starting point to add new features to BPMN or to enrich it with new capabilities or symbols. We have already done some successful experiments integrating BPMN with Privacy Policies expressed using P3P (see Chap. 6) or adding some time/cost metrics.

3.5 BPeX vs. BPMN 2.0 Proposals

In this section we examine the differences between the two proposals for BPMN 2.0 and BPeX in detail. Wherever it is possible, we explain why those differences exist, though in many cases the only real explanation is that two different design teams inevitably come up with different answers to the same problem.

At this point it is important to provide a quick survey on the BPMN-related notations, standards and specifications to better understand the goals detailed inside the OMG BPMN 2.0 RFP.

We currently have to deal with a large set of business processes modeling notations, either standard or emerging proposals. Figure 3.8 was already presented

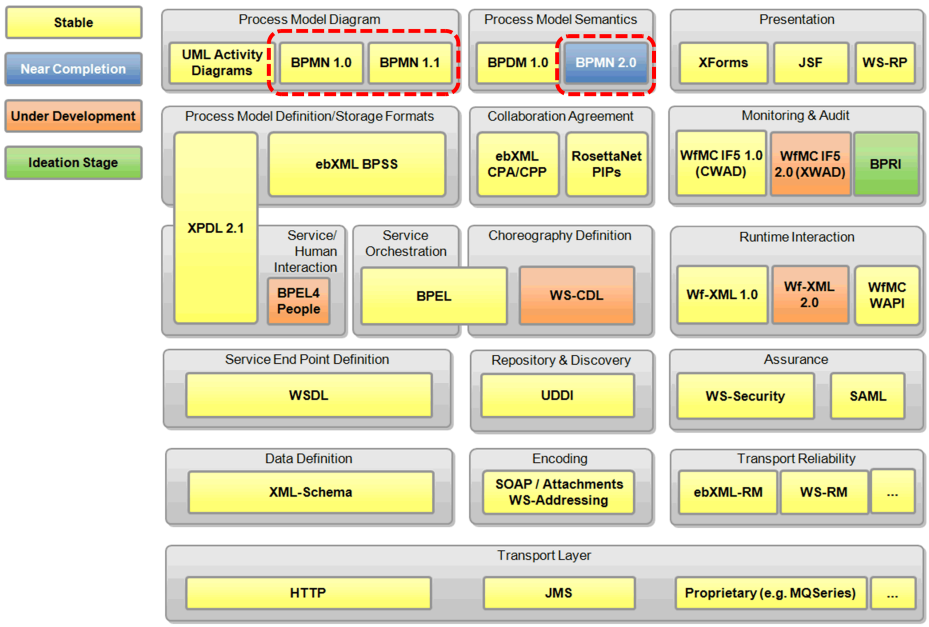


Figure 3.8: The BPMN placement with respect to the other business processes related standards

in Chapter 1 to emphasize the BPMN placement with respect to the other standards. As we have already shown in the beginning of this chapter, BPMN and the related activity came from several different efforts. In some cases the original proposals do not stop their developing, spinning off other parallel projects. The main purpose of the BPMN 2.0 RFP is to merge together some relevant proposals of such parallel projects in a single notation to be standardized, which will be adopted as a de-facto standard notation.

Focusing on BPMN we have to deal with a lot of different specifications, each one with its own scopes and application domain. Now we present some of them picked out from the whole by their closer relationships with the BPMN 2.0 proposals. The *Business Process Maturity Model* (BPMM) [8, 22] describes an evolutionary improvement path that guides organizations moving from inconsistent

processes to disciplined processes. It provides a reference model for appraising processes within the enterprise. BPMM provides a detailed roadmap for business process improvement. The *Business Process Definition Metamodel* (BPDM) [50] is a rigorous metamodel invented by OMG as a foundation for BPMN. BPDM provides the bridge between business-friendly BPMN diagrams and the software tools that automate and coordinate business processes across organizations. In other words, BPDM could be used to manage processes using different approaches (mainly, structured and unstructured models, as in the case of UML and BPMN) along with to gain the capability to provide a BPMN serialization, competing to XPDL and other solutions. BPDM also supports orchestration and choreography – two complementary views of processes. BPDM was designed to supplement BPMN with a formal metamodel of its modeling constructs, even if it was deemed too hard by many people. The *Business Motivation Model* (BMM) [5] provides a structure for documenting, communicating and managing the key elements of the business and their interrelated purposes in an integrated way. BMM gives a taxonomy to specify goals and objectives of organizational activities and structures. At last, the *Semantics of Business Vocabulary and Business Rules* (SBVR) [51] specifications provide the means to precisely define business terminology, formally and unambiguously specifying each definition in terms of other definitions in the vocabulary. It provides a formally defined taxonomy to describe elementary business operations and rules. It has an UML metamodel. An example of SBVR rules expressed using SBVR Structured English[4] can be seen in Fig. 3.9.

Modality Type	Modality Operator	Example in SBVR Structured English
Structural (alethic)	necessity	An order always *has* exactly one customer.
	non-necessity	It is not necessary that an order *has* a customer.
	possibility	It is possible that an order *has* more than one line item.
	impossibility	An order never *has* more than one customer.
	contingency	It is possible but not necessary that an order *has* a customer.
Behavioral (deontic)	obligation	Each order must *be processed within* one business day.
	non-obligation	It is not obligatory that a customer representative *approve* a credit.
	permission	A customer representative may *approve* a credit.
	prohibition	A clerk must not *change* the terms and conditions.
	optionality	It is permitted but not obligatory that a customer representative *approve* a credit.

Figure 3.9: An SBVR rules example

More detailed information about these specifications can be found on the OMG

[4]Examples from SBVR specification, Annex E, EU-Rent case study

affiliate BPMRoundtable website[5]. Moreover, we already discussed in the previous sections about BPEL and XPDL.

All that we briefly presented here (plus some other projects and proposals) represents the as-is-now situation. In June 2007 OMG published a Request for Proposal for BPMN 2.0 (submissions closed in February 2008) to bring order among the above mentioned artifacts. As a matter of fact, the BPMN 2.0 RFP calls for: *submissions that reconcile the Business Process Modeling Notation (BPMN) and the Business Process Definition Metamodel (BPDM) to specify a single language with metamodel, graphical notation and interchange format.*

Moreover, the RFP calls for:

- A single specification, entitled Business Process Model and Notation (BPMN 2.0), that defines the notation, meta-model and interchange format with a modified name that preserves the "BPMN" brand.
- Extension of BPMN notation to address BPDM concepts.
- Changes which are required to reconcile BPMN and BPDM to a single, consistent language.
- The ability to exchange business process models and their diagram layouts among process modeling tools preserving semantic integrity.
- Enhancements in BPMN's ability to:
  - Model orchestrations and choreographies as stand-alone or integrated models.
  - Support the display and interchange of different perspectives on a model that allows a user to focus on specific concerns

There are also some other requirements which on one side restrict the autonomy of the proposals (e.g., *"Submitters are encouraged to express models using OMG modeling languages"*, and also *"Proposals shall reuse existing OMG and other standard specifications in preference to defining new models to specify similar functionality"*), but on the other side fix some more detailed constraints useful to have a consistent submission. As it stands out, it is required that the new standard include some way of representing the conceptual architecture of the model.

Finally, the RFP lists some other specifications (published by OMG but also by other subjects) which have to be considered for the proposals. Some of these are:

- OMG
  - Meta Object Facility Specification
  - MOF 2.0 Versioning
  - XML Metadata Interchange (XMI) Specification version 2.1
  - Software Process Engineering Metamodel (SPEM)
  - Unified Modeling Language (UML 2.1.1)

[5] `www.bpmroundtable.org/101807_BPM_Fact_Sheet.pdf`

- WfMC
  - Workflow Reference Model
  - Terminology & Glossary
  - Workflow Process Definition Interchange
  - XML Process Definition Language (XPDL) version 2.0
- W3 Consortium
  - Web Services Definition Language (WSDL 1.1)
  - Web Services Choreography Definition Language (WS-CDL)
- OASIS
  - ebXML Business Process (ebBP) 2.0.4
  - Business Process Execution Language (BPEL)

This summary is useful to understand some guidelines behind the submitted proposals. The two submission groups, which we will soon briefly introduce, tried to solve the same problems adopting different solutions abiding by the RFP constraints and requirements. In 2008 we became OMG academic members, joining also the BPMN 2.0 Working Group. We accessed BPMN 2.0 subsequent submissions but we can refer in this work only to the last one published by OMG – November 2008 – and which can be freely accessed by everyone. We will take some sentences from the two proposals to compare them on certain distinguishing characteristics, as for choreographies definitions. BPMN 1.1 does not provide an explicit Choreographies definition, even if it is possible using Message Flows together with other BPMN elements to represent interactions between business processes. So, the OMG RFP calls for a clear and formal definition of business processes choreographies. We will quote the Choreography definitions given by both BIOS and BPMN-S proposals to show how the two groups provide different descriptions of the same topic.

BIOS proposal. The initials "BIOS" stand for BEA/IBM/Oracle/SAP, which was the original companies group that initially submitted this proposal. Later on, the group changed because Oracle bought BEA and Unisys joined the group. The last public submission[6] is signed by IBM, Oracle, SAP and Unisys as formal submitting members. In late 2008 we joined too this submission group as OMG academic members. Our goal is to provide suggestions for the modification of the proposal in order to overcome the deficiencies we found out, as explained in Section 3.5.1. We remark that this critical analysis has been possible by our previous work on BPeX.

According to the submission proposal, BIOS BPMN 2.0 specifications proposal extends the scope and capabilities of the BPMN 1.1 in several areas:

- they formalize the execution semantics for all BPMN elements

[6] `http://www.omg.org/cgi-bin/doc?bmi/08-11-01`

- they define an extensibility mechanism for both process model extensions and graphical extensions
- they refine event composition and correlation
- they extend the definition of human interactions
- they define a choreography model

This specifications also resolves known BPMN 1.1 inconsistencies and ambiguities. What follows is an excerpt from the text to show an example of the specifications format. It concerns the business processes choreographies definition. For the full text access refer to the OMG website. The structure of the document traces out the old BPMN specifications style. This group chose as new name for BPMN standard the initials "BPMN 2.0", according to the OMG RFP which requires to provide a new name which saves the BPMN brand.

> **Choreographies**. A self-contained Choreography (no Pools or Orchestration) is a definition of the expected behavior, basically a procedural contract, between interacting Participants. While a normal Process exists within a Pool, a Choreography exists between Pools (or Participants). The Choreography Process looks similar to a private Business Process since it consists of a network of Activities, Events, and Gateways (see Fig. 3.10). However, a Choreography is different in that the Activities is an interaction that represents a set (1 or more) of message exchanges, which involves two (2) or more Participants. In addition, unlike a normal Process, there is no central controller, responsible entity or observer of the Process.

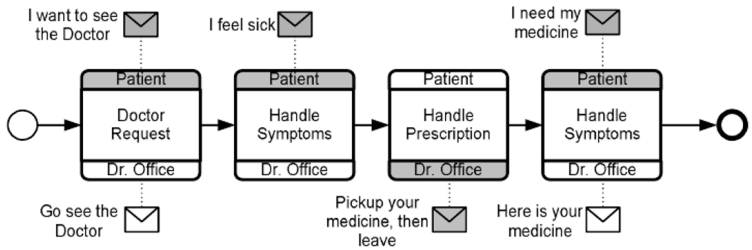


Figure 3.10: An example of a Choreography

BPMN-S proposal. This is the "BPMN for Services (BPMN-S) submission team", which is composed by: Adaptive, Axway Software, Hewlett-Packard, Lombardi Software, MEGA International, Troux Technologies, Unisys. They used a different approach with respect to the BIOS proposal. Reading from the last submitted text[7], BPMN 2.0 is designed to support, among others, the following functionalities:

[7] http://www.omg.org/cgi-bin/doc?bmi/08-11-07

- Business level modeling
- Checking compliance to choreography
- Checking compliance to rules
- Process taxonomies
- Execution interoperability
- Runtime management extensibility
- Language extensibility
- Model reusability
- Bridging business and information technology

These capabilities are enabled by modeling occurrences of processes as well as process definitions. As for the previous example, also in this case we provide an example concerning the business processes choreographies definition. The structure of these specifications is completely different from the 'classical' BPMN specifications style, tracing out the WfMC documents style. This group adopted a completely different approach to define the new version of the BPMN standard. In the following excerpt comes out how this group chose to implement occurrences of processes to equip BPMN with innovative features. This group chose as new name for BPMN standard the initials "BPMN2".

> **Choreographies** and messages also happen between particular times, and occurrence apply to them also. Each occurrence of a message starts when the message is sent and ends when it is received. Choreographies order messages in time just as processes order steps or activities. For example, Figure 3.11 has a choreography definition at the top, and three occurrences of the choreography at the bottom (the time line also shows occurrences of the messages). Each occurrence is unique and happens between particular times. The two occurrences on the lower left follow the Contracting choreography model, while the one on the right does not, because the occurrence of the Requirements message happens after the Quote message. The left occurrences show it is not necessary for the Quote message to be sent immediately after Requirements message is received, which is critical to choreographies, because the internal processes of the partners occur in between messages. Choreography modeling is not possible in token-based or other virtual machine semantics, because machine semantics gives a strict series of steps to follow. For example, when the Requirements message arrives at the Contractor, a token-based semantics would immediately have the Quote message sent to the Client. It would not allow the internal process of the Client to happen in between. In general, operational semantics cannot support choreography modeling.

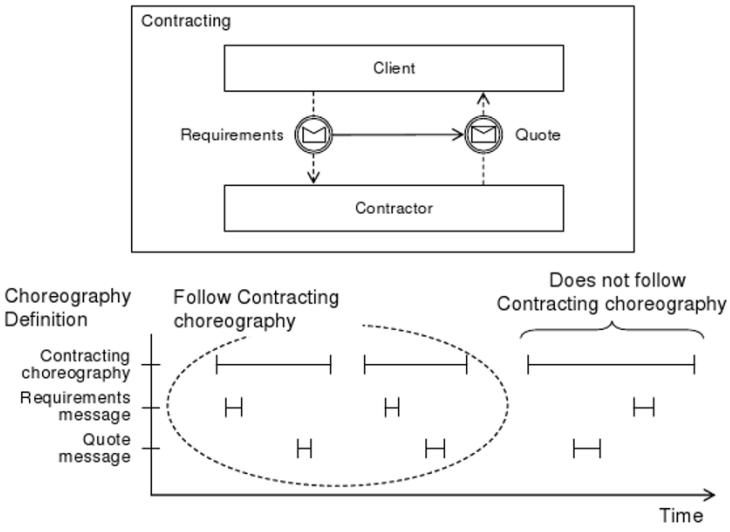


Figure 3.11: Choreography Models and Occurrences

3.5.1 A Critique of BPMN 2.0 Proposal

Having presented the state-of-the-art of BPMN standards and proposals, now we expose our contribution in the BPMN 2.0 submission process. Thanks to some works we discussed during the last years, we started collecting issues on BIOS BPMN proposal. Recently, we were invited to become OMG member to be able to actively collaborate with this proposal. Therefore, we have analyzed in depth the BIOS proposal and in the following we report our comments and some issues. The submission version we refer to is the initial submission published in February 2008. We insert when significant several notes about how BPeX can be used (under some assumptions) to better model particular BPMN facets. As we already mentioned, our contribution will be formalized in the forthcoming BIOS BPMN 2.0 submission.

The BPMN conceptual model. The provided metamodel is very complex. It seems very useful to have multiple concentric layers to permit extensibility mechanism without the need to modify the BPMN core elements definition. It is not immediate to establish a direct correspondence between the metamodel classes and the BPMN structural elements. It seems to be a type-centric metamodel definition and not an element-centric metamodel definition. This certainly makes easier to extend the model but it introduces more complexity and some attributes overloading. Looking at the classes names and trying to build a subset of those classes needed to define an element it is possible to realize how complex it seems.

With BPeX we tried to simplify the elements definition, building the metamodel from scratch starting from the original BPMN metamodel and its elements definitions. We structured our model defining inheritance and generalization correlations between elements, as we discussed in Chap. 2.

BPMN methodologies. Methodologies are still missing. There is a lack of a design methodology to support modelers define their processes following a finite list of steps. Also methodologies to analyze and evaluate models are missing.

We will present in Sect. 4.1 our business processes design methodology, borrowing some basic concepts from DBMS domain and applying them to business processes modeling domain. We will provide one example process to support our design methodology proposal.

User management. User management has been improved in this submission. There were introduced new concepts and definitions like Human Interactions, People Group and People Assignment. It is not completely clear how these concepts could be used in practice because a lack of examples and more detailed explanations. User management is still framed to the description of users interactions as parts of the process to be represented.

In Sect. 5.4 we will introduce in BPeX the possibility to consider also people accessing the diagram displaying only the elements subset for which they have access permissions. We also add the capability to describe a full set of access control policies for real users who need to update the business process diagram.

Choreographies. As we mentioned before, BPMN 1.1 does not provide an explicit Choreographies definition, even if it is possible using Message Flows together with other BPMN elements to represent interactions between business processes. As requested by the OMG RFP this proposal introduce a formal definition of choreographies. However, there are some details which could make the new BPMN model too complex. For instance, BIOS proposal provide the possibility to define different types of Sub-Processes inside Choreographies elements. The basic Sub-Process element could be useful to better represent Choreographies Tasks at higher levels of abstraction, but the introduction of looping or multi-instance types might complicates too much the model.

Gateways definition. There are two main issues inside the Gateways definition. First, Gateways could have no Inputs or Outputs Flows. It is not clear when this condition could happen, but assuming to deal with Gateways without Input or Output Flows, they could be intended as starting or ending points of one process in default of other information. Second, the definition of Inclusive Gateways is confusing, because, unlike the BPMN 1.1 specifications, the BIOS proposal introduces the concepts of *dominator* and *post-dominator* without giving any sort of explanation.

Events definition. Also for Events definitions there are two main issues. The BPMN specifications state that if one Start (or End) Event is present, then there MUST be also an End (Start) Event. However, in BIOS proposal there are statements asserting that End (Start) Events could be omitted (or they are implicit) even if a Start (End) Event is explicitly declared. This seems to be a contradiction, unless in this case the word "implicit" used in the proposal means "not graphically represented", assuming that missing Events are declared somewhere (e.g., inside the XML serialization). Moreover, a new Event is introduced: the

Escalation Event. But BIOS proposal does not specify how it has to be graphically represented. Also the Event type definition – whether it is a Start, an Intermediate or an End Event – is unknown. Its attributes list is provided, but a description of its behavior is missing.

Minor considerations. Inside this last paragraph we want to list some other minor issues we found in BIOS submission.

- There is a new Task type definition: the Null Task which performs no actions. It was introduced only to have a better and likely arranged graphical representation of the model. It seems not to be so necessary for the process modeling.
- InputSets and OutputSets attributes are defined as mandatory, but leaving the possibility to define them with empty values. It could be a better choice to define them (and other attributes defined in the same way) as optional attributes. Otherwise There exists the risk to generate models with a lot of unnecessary elements. Another example concerns the Collaboration definition. One Collaboration MUST hold at least 2 Pools, but later on in the same paragraph Collaborations are described as they could contain only one Pool.
- There are sometimes redundant connections between elements. As an example, DataInputs and DataOutputs are connected to Activities and Events through the Extensibility Mechanism even if a relationship between DataObjects and Activities and Events was already defined using the Association connector.
- Most elements definitions are positioned apart from the elements first usage, making the proposal more difficult to understand.
- BIOS proposal aims to give Data Management more importance. It defines new DataObjects related concepts like Lifecycle and Scope as well as Data Transformations. Unfortunately, there is no mention of what "Data Transformation" means. Moreover, it seems that DataObjects are no more parts of native BPMN Artifacts but their definition is missing. It is not clear if they were defined as stand-alone objects or as a new elements family.

Chapter 4

Business Process Modeling Methodology

Having a business processes notation and a corresponding conceptual model is not enough when dealing with the modeling of large, complex business processes. Following tradition, we introduce a BP modeling methodology to support users represent business processes.

Very often in real scenarios business processes are described using a more or less formalized languages. The visual support is often missing and thus having a graphical tool to aid users modeling their processes is a first, necessary goal to be achieved. During the last few years there were a lot of different purposes to accomplish this requirement, e.g. UML, PN, BPMN, Workflows, EPC, but also spreadsheets to organize data and processes documentation. In the meanwhile there were purposes to serialize and to represent these models in a more formal and suitable way, in some cases using the same notations listed above. The need of having a serialized version of the process to be able to perform some kind of analysis or validation or to exchange processes between different environments or software architectures, addresses most of the research efforts. Our proposal fill a wide gap in the business processes modeling scenario, since there is no mention about BP Design and Modeling methodologies inside the most recent specifications published or submitted to be published (as for example in the case of BPMN 2.0 proposals). Nonetheless, developing a modeling methodology is very important since for the next five years academics and companies suggest as ever more important to develop methodologies [40].

In this chapter we present a BPMN design methodology. By a business process design methodology we mean a set of transformation rules specifying how a business process element can be modeled and a set of (possibly semi-formal) rules addressing how such transformation rules should be applied in order to derive a correct and complete business process, with respect to the initial specifications. We will use as a reference our BPeX conceptual model for BPMN and its XML serialization instead of the BPMN 1.1 UML metamodel (the newest published model – we will take a closer look on BPMN 2.0 proposals later on in this work).

There are many works addressing Business Process Reengineering (BPR) (also

known as Business Process Redesign). BPR are synonyms to indicate a systematic, disciplined five-steps approach that critically examines, rethink, redesign, and implements the redesigned processes of an organization supporting the passage from the 'as-is' modeling towards the 'to-be' modeling [10, 9, 25]. Davenport and Short [10] define BPR as "the analysis and design of workflows and processes within and between organizations". All the works pertaining BPR discuss about conditions and motivations to reengineer a process or to redesign a process, presenting also different approaches to solve the problem. However, at our best knowledge, there are no works addressing clearly the problem of have the definition of a clear business process design methodology. That is: a set of rules stating how to start from the specifications (e.g., a natural language description) and iteratively define a representation of business processes. A different, if somewhat related, task is described in works dealing with methodologies for checking the compliance of a generic business process with respect to some well defined set of rules and properties, such in BPMN specifications. Given the large number of rules and their frequency of change, manual compliance checking can become a time-consuming task [3]. Automated compliance checking of process activities and their ordering is an alternative whenever business processes and compliance rules are described in a formal way. We believe that using our methodology in order to represent business processes in BPMN significantly reduces the need for future redesign phases.

Establishing what are the properties that a "good" business process should have is a hard task. Here we informally mention some properties that will be more thoroughly discussed in the following. A good process model must be correct, relevant and economical [67]. It is correct if it has a correct syntax (proper use of the modeling method) and a correct semantics (an accurate representation of the processes behavior). A process model is relevant if inside the model everything is relevant and if everything inside the model is relevant. At last, a process model is economical if the effort spent to produce the model supporting documentation is reasonable if compared with the complexity of the model use.

4.1 Business Processes Design Methodology

To present our BPMN design methodology we start with an example. The example models a process describing a generic customer-supplier scenario. The process does not model all the aspects of a real scenario (e.g., there is no the description of the actions performed by the Customer if the payment procedure fails) but it is complete enough and it employs enough elements to be used for the aims of this section. The example process is described using natural language as follows:

> *Everytime someone wants to buy a new MP3 player he has to go to the nearest open store. The customer waits to be served. If his waiting exceeds a reasonable time (e.g., 10 minutes) he leaves the store without buying anything. On the contrary, he asks the sale assistant for an MP3 player. The sale assistant takes the order and forwards it to the store warehouse clerk who looks for the requested object. Meanwhile, the sale assistant proceeds with the payment procedure. The customer pays for the MP3 player he requested. If a problem with the payment process occurs, the sale assistant stops immediately the warehouseman who discards the order and finishes his task. Otherwise, the cashier sends a message to the warehouseman to confirm the order. Finally, the customer withdraws his MP3 player from the warehouse.*

We propose a *three-phases* methodology to support modelers design processes starting from natural language specifications. Each *Phase* is composed by a set of *Rules* which act as guidelines for the modeler to obtain a well-formed and valid business process model starting from natural language specifications. Here, *valid* and *well-formed* mean having result diagrams which are compliant to BPMN specifications under several aspects as, for example, graphical representation, syntax and semantics, attribute declarations. We deal more in detail with this concept in the following. It is important to state that to follow these rules starting from one process description in natural language could produce different valid diagrams – assuming that the resulting diagrams fulfill the requirements, correctly describing the original process aspects and behavior. This is because there are several ways to model a process depending on different factors, not least the modelers diverse skills.

Instead of starting from scratch, to develop the BPMN design methodology we chose to start with a well-known modeling methodology (the ER modeling methodology for databases, but also the more general ANSI definition[1]), adapting it to our needs. The significance of this approach, according to ANSI, is that it allows the three phases to be relatively independent of each other. Hence, this design methodology could be applied independently from a BPMN specific element set, from the underlying model or, even, from the chosen serialization language (with the adequate changes if needed). The BPMN design methodology is defined as follows:

Phase 1: Conceptual Modeling. A conceptual data model is a map of concepts and their relationships. This model describes the structure of an

[1] American National Standards Institute. 1975. ANSI/X3/SPARC Study Group on Data Base Management Systems; Interim Report. FDT (Bulletin of ACM SIGMOD) 7:2.

organization and states a series of assertions about its behavior. Specifically, it describes the relevant concepts to an organization (entities, classes), about which it is inclined to collect information, attributes and associations among the above entities (relationships). This phase includes the *Design Rules*, through which it is possible to reason on the natural language specifications of the process and to start model an high-level business process diagram. Modelers are free to assign arbitrary labels to object and to use BPMN elements as they could be useful to have a diagram reflecting the textual description.

Phase 2: Logical Modeling. During this phase the diagram obtained as result from the first phase is analyzed following some *Reduction Rules* and *Refinement Rules*. These two sets of rules could be executed repeatedly until the BPMN diagram turns into a relevant and economical model. Here modelers have to use the appropriate BPMN elements trying to substitute the simplest elements where possible. Moreover, all the element attributes and properties (or, at least, those which are mandatory for the BPMN specifications) should be declared.

Phase 3: Physical Modeling. In this phase the serialization of the second phase result diagram in XML code is accomplished. To have a well-formed and valid diagram in BPMN it is a necessary condition to be able to create an XML description of the process.

In the next few sections we illustrate the methodology one *phase* at a time using the example process we introduced before. For each *Rule* we give also some informal guidance on when it is necessary to apply each particular *Rule*.

4.1.1 Phase 1: Conceptual Modeling

Natural language specifications very often are the starting point of the design phase. Most enterprises have tons of papers describing internal processes and their relationships with other participants (e.g., customers, suppliers). It is really difficult to understand the behavior of a process and only few employees can accomplish this task successfully. Few attempts to adopt a standardized lexicon for writing specifications have been made (e.g., SBVR [51], which has been presented more in details in Chap. 3). But up to now they are not very well known. Thus, the lexical analysis upon which this first phase is centered is still a first necessary step. The output of this phase will be a first version of the process modeled with BPMN, using especially the BPMN core elements set [68]. The goal is to obtain one correct, complete and compliant graphical representation of the process to be still refined and upon which infer some properties. We report again the example process text for explanation convenience.

> *Everytime someone wants to buy a new MP3 player he has to go to the nearest open store. The customer waits to be served. If his waiting exceeds a reasonable time (e.g., 10 minutes) he leaves the store without buying anything. On the contrary, he asks the sale assistant for an MP3 player. The sale assistant takes the order and forwards it to the*

store warehouse clerk who looks for the requested object. Meanwhile, the sale assistant proceeds with the payment procedure. The customer pays for the MP3 player he requested. If a problem with the payment process occurs, the sale assistant stops immediately the warehouseman who discards the order and finishes his task. Otherwise, the cashier sends a message to the warehouseman to confirm the order. Finally, the customer withdraws his MP3 player from the warehouse.

Rule 1.1: Participants identification. Participants are all the actors, services, people, employees involved in the Process execution. Participants are those entities or roles which perform actions, activities and tasks. Every Participant in BPMN is represented with a `Pool` (from BPMN specification: *"A Pool represents a Participant"*). Thus, for each Participant a different Pool must be drawn. The Pool name will be the same of the Participant's (Fig. 4.1). If one ore more actors are in hierarchical relationship with one other, it is safe to represent them partitioning the Pool with necessary Lanes.

Very often, participants can be identified as the actors of the process. Looking at the example, we highlight them with boxes.

Everytime *someone* *wants to buy a new MP3 player he has to go to the nearest open* *store*. *The* *customer* *waits to be served. If his waiting exceeds a reasonable time (e.g., 10 minutes) he leaves the store without buying anything. On the contrary, he asks the* *sale assistant* *for an MP3 player. The* *sale assistant* *takes the order and forwards it to the store* *warehouse clerk* *who looks for the requested object. Meanwhile, the* *sale assistant* *proceeds with the payment procedure. The* *customer* *pays for the MP3 player he requested. If a problem with the payment process occurs, the* *sale assistant* *stops immediately the* *warehouseman* *who discards the order and finishes his task. Otherwise, the* *cashier* *sends a message to the* *warehouseman* *to confirm the order. Finally, the* *customer* *withdraws his MP3 player from the* *warehouse*.

There are no semantic rules to determine when two or more actors' names are the same. One well described process should follows a fixed and shared taxonomy, at least inside the company boundaries. Without a predefined taxonomy or glossary, we have to recognize synonyms, technical or standard terms. In the example process we can easily found four different actors: one customer (someone, customer), one store, one sale assistant (sale assistant, cashier) and one warehouseman (warehouse clerk, warehouseman, warehouse). We can infer from the context that the warehouseman and the sale assistant work for the store and, thus, we can conclude that we have to deal with two processes: one for the customer and one for the shop. The shop will have two lanes, one for each employee. The result of applying this first rule can be seen in Fig. 4.1.

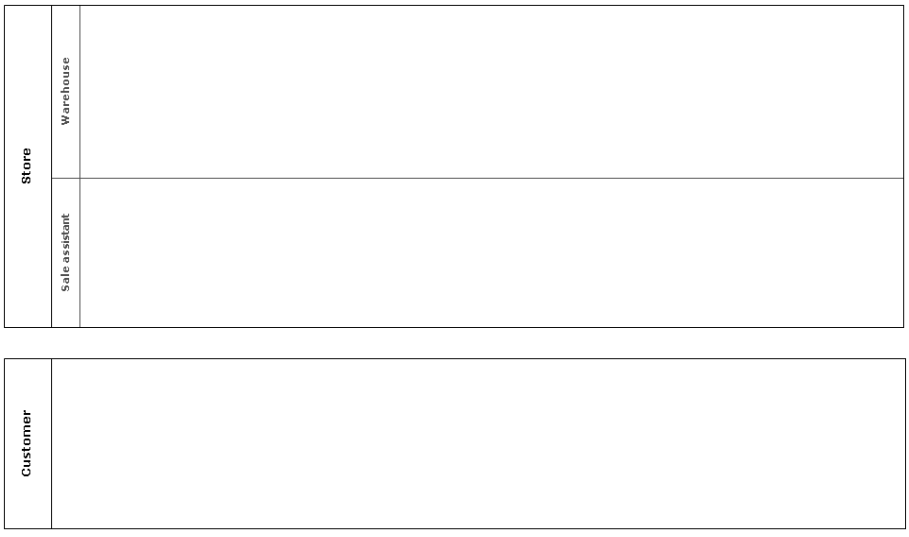


Figure 4.1: Rule 1.1: Participants identification and graphical representation

Rule 1.2: Activities identification. For each Participant, the main activities set is to be identified. If an action taken from one of the Participants has a simple structure and does not have relevant associated properties, it can be represented with an Activity Task. During the first phase, it is possible to define all tasks as of type *None*. If one action performed by one of the Participants is a complex action, which could be specified more in details, it can be represented using Sub-Processes.

Note in this rule of the desired abstraction level to be represented the process. It is necessary to make a semantic analysis of the actions performed by the Participants to state if an action has to become a Task or a Sub-Process. Typically, atomic actions are to be represented with Tasks, while complex actions with Sub-Processes. Modeling a Sub-Process does not necessary mean that it is required to specify also the content of the process. BPMN permits to represent collapsed Sub-Processes making feasible to indicate that an activity is composed from simple tasks to a whole process. Sometimes the text description of the activities gives no detailed information pertaining the behavior or the content of such elements. It is likewise possible to model them. During the second phase there will be the chance to substitute Sub-Processes with Tasks and the other way round. Activities are the most expensive elements inside a process, and thus this step is very important. Model different layers of granularity could be useful to represent also different views of the same process. In the next chapter we will deal with the definition of different views on a process.

One guidance is to define all the actions as Tasks, refining the diagram during the second phase. This is because is more simple to group activities together obtaining a Sub-Process (e.g., an Ad-Hoc Sub-Process) than extracting information form a Sub-Process splitting it into several different Tasks. The latter way is

not recommended because as consequence the general complexity of the process rapidly increases. At last, it is important to give attention also to distinguish between something that "happens" during the process (Events) and a work that participants perform (Activities).

We report another time the example process text with in this case all the activities framed.

> *Everytime someone wants to buy a new MP3 player he has to* [*go*] *to the nearest open store. The customer* [*waits*] *to be served. If his waiting exceeds a reasonable time (e.g., 10 minutes) he* [*leaves*] *the store without buying anything. On the contrary, he* [*asks*] *the sale assistant for an MP3 player. The sale assistant* [*takes*] *the order and* [*forwards*] *it to the store warehouse clerk who* [*looks*] *for the requested object. Meanwhile, the sale assistant* [*proceeds*] *with the payment procedure. The customer* [*pays*] *for the MP3 player he requested. If a problem with the payment process occurs, the sale assistant* [*stops*] *immediately the warehouseman who* [*discards*] *the order and* [*finishes*] *his task. Otherwise, the cashier sends a message to the warehouseman to confirm the order. Finally, the customer* [*withdraws*] *his MP3 player from the warehouse.*

Participants		**Activities**
Store	*(Sale assistant)*	take, forward, proceed, stop
	(Warehouseman)	look, discard, finish
Customer		go, wait, leave, ask, pay, withdraw

Table 4.1: Participants with related Activities

The Tab. 4.1 shows the matches between participants and activities. This is an important step if a modeler has to deal with very big and complex processes. This table helps users to correctly position the activities inside the Pools and Lanes boundaries, as in Fig. 4.2.

Rule 1.3: Events identification. An event is something that "happens" during the course of a process, affecting its flow. Depending on when an event occurs, events could be classified as Start Events (when they occurs at the beginning of the process, causing the start of a process), End Events (when they indicate the end of a process) or Intermediate Events (when they occur between the start and the end of a process). They are represented in BPMN with respectively Start, End and Intermediate Events.

Without using a formalized language it could be difficult to distinguish between some kind of activities and events. As an example, the Sale Assistant which sends a message to Warehouseman can be interpreted as an Activity or as an Event (as we do). Actually, there could be activities that may be modeled

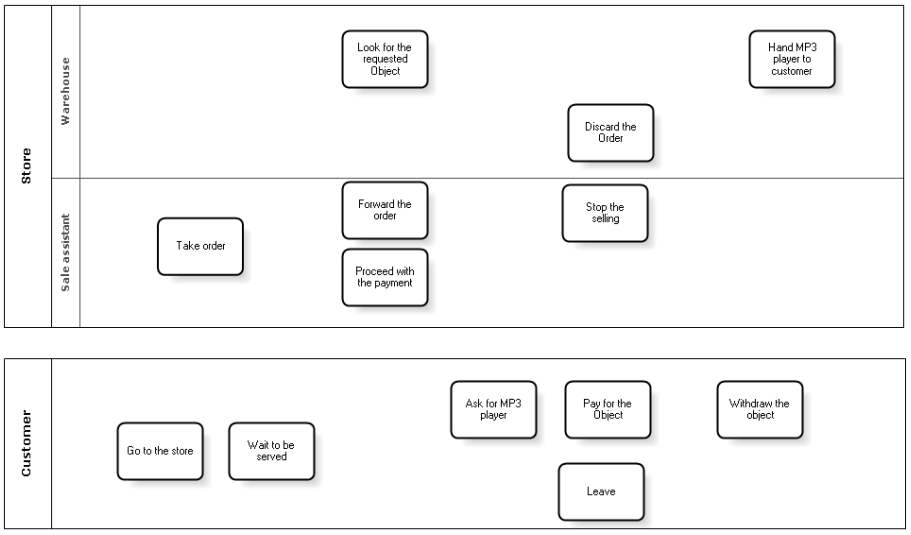


Figure 4.2: Rule 1.2: The example process populated with Tasks

as events, saving the number of activities (more complex to model and to serialize) and simplifying the diagram. Moreover, some events are not clearly specified inside the process description. This could be a problem in the case of Start or End events. BPMN specifications provide the possibility not to draw Start or End Events starting and ending the process with Tasks (but if at least one Start (End) Event is designed, there must be at least a corresponding End (Start) Event for the same process). Similarly to the previous rule, the Tab. 4.2 reports the events identified in the example process. Figure 4.3 shows the process as it should look like having applied the first three rules. Note that for a sake of clarity we specified also the empty Start and End Events for both the processes where needed. Moreover, the more suitable element type relative to the *'occurs'* event should be an Intermediate Event, but we prefer to use in this case an End Event because the Sale Assistant's workflow ends with such an event.

Participants		**Events**
Store	*(Sale assistant)*	*a problem with the payment* occurs, sends *a message*
	(Warehouseman)	stops *immediately*, finish *the task*
Customer		*if his waiting* exceeds

Table 4.2: Participants with related Events

Rule 1.4: Choices identification. It is possible to identify choices everytime a single flow could be splitted in more than one different path. Choices are often introduced by conditional or boolean expressions, like, for example, *if... then [... else]*, *while/meanwhile*, *or/otherwise*, and so forth. Choices

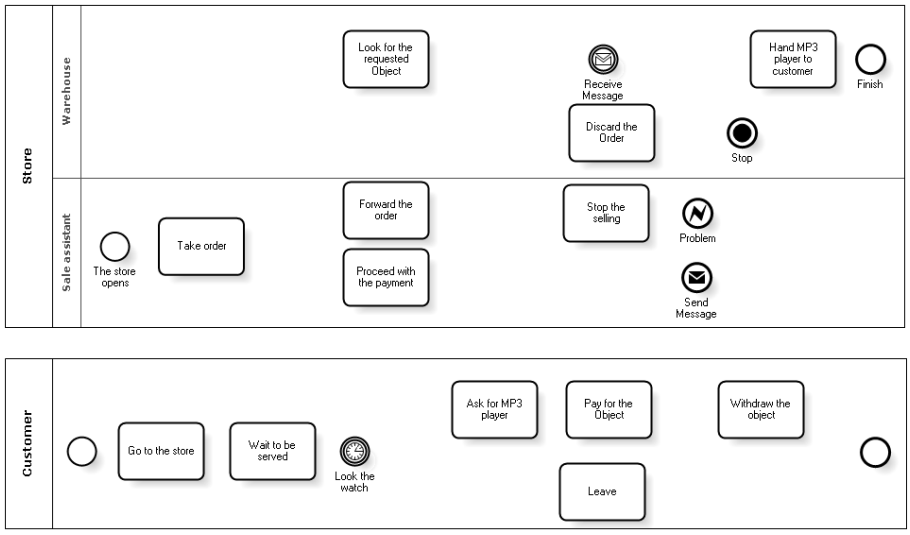


Figure 4.3: Rule 1.3: The example process after the Events design

could also be identified through conditional form of such verbs, like sentences containing the verbs *can*, *may* or the conditional forms *could*, *should*, *would*, and so on. Some of these constructs suggest which type of choices we are dealing with. For example, terms like *while/meanwhile* clearly indicate the possibility to have parallel flows (activities executed at the same time) while terms like *or/otherwise* suggest the use of alternate paths. Choices in BPMN are represented using Gateways. Gateways affect the flow of the process forking, branching, joining or merging different paths.

We can have inside a process different kinds of Gateways. There could be the need to split a flow in multiple parallel flows or to choice which path to follow depending on the result of a condition or, even, exceptions management if particular conditions occur. Every splitting Gateway must have a merging Gateway (which may be implicit) or End Events to end all the paths. Simply follow each outgoing path until its termination is reached. BPMN specifications provide the *Token* notion to simulate the flows. There should not be left processes with non-terminating paths. Also loops could imply some non-termination conditions. Sometimes conditions are implicitly defined as in the case of our example in which the Warehouseman has to change his work flow if a message arrives from the Sale Assistant. Framed words in the following text, Tab. 4.3 and Fig. 4.4 refer to the example process showing the found gateways. We framed with a double box the implicit Gateway.

Everytime someone wants to buy a new MP3 player he has to go to the nearest open store. The customer waits to be served. $\boxed{\boxed{\textit{If}}}$ *his waiting exceeds a reasonable time (e.g., 10 minutes) he leaves the store without buying anything.* $\boxed{\boxed{\textit{On the contrary}}}$*, he asks the sale as-*

sistant for an MP3 player. The sale assistant takes the order and forwards it to the store warehouse clerk who looks for the requested object. *Meanwhile*, *the sale assistant proceeds with the payment procedure. The customer pays for the MP3 player he requested.* *If* *a problem with the payment process occurs, the sale assistant stops immediately the warehouseman* *who discards* *the order and finishes his task.* *Otherwise*, *the cashier sends a message to the warehouseman to confirm the order. Finally, the customer withdraws his MP3 player from the warehouse.*

Participants	**Gateways**
Store *(Sale assistant)*	Meanwhile (Parallel Gateway), If/Otherwise (Exclusive Gateway)
(Warehouseman)	'who discards' (implicit Exclusive Gateway)
Customer	If/On the contrary (Exclusive Gateway)

Table 4.3: Participants with related Gateways

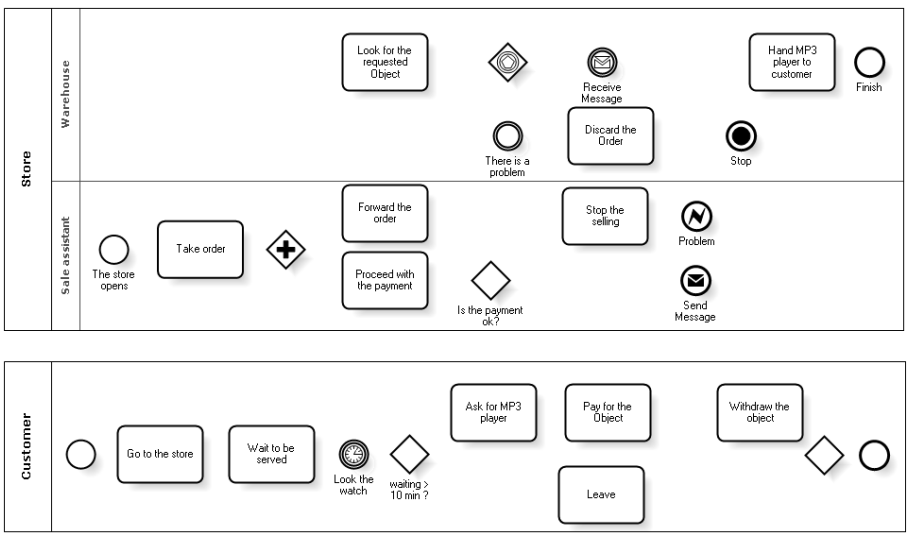


Figure 4.4: Rule 1.4: The example process with the Gateways added

Rule 1.5: Adding Relationships. There are two different kind of relationships affecting flows: Sequence Flow and Message Flows. The easiest way to distinguish among them is to consider Sequence Flows as the representation of the order that activities will be performed **within** a process, while Message Flows are used to show the messages exchanged **between** different processes.

- **Rule 1.5.1: Sequence Flows.** Following the order given by the process specifications, connect all the activities, events and gateways, beginning from Start Events and finishing with End Events. This step is to be performed for every participant separately.
- **Rule 1.5.2: Message Flows.** Take a look to the actions which involve two or more participants. Every information exchanged between different participants has to be represented with a Message Flow arising from the sender towards the recipient. Notice that the content of the Message exchanged could be a textual message or a whatever object, a paper document as well as an email.

It is a non-trivial work (especially if we have to deal with complex processes having an huge amount of elements) to list all the Sequence Flows needed to connect all the diagram shapes. It is useful, instead, to report in a table the list of the relationships established among processes, as it is shown in Tab. 4.4 for the running example. Wherever possible, consider to apply workflow patterns [55], BPMN provided patterns or other well-known flow patterns. If the company has some tested and used patterns, consider the possibility to build a summary of all the good-working patterns.

From	**To**	**Message**
Customer	Store	Order
Customer	Store	Payment
Store	Customer	MP3 Player

Table 4.4: Message Flows for the running example process

Rule 1.6: Documentation of the processes. If the process description provides some other information which has not been yet represented using the BPMN core elements set, it is possible to add them using BPMN Artifacts.

First Phase Concluding Remarks. In Fig. 4.5 it is possible to see the example process modeled with BPMN as it should appear at the end of the first phase (a bigger version of the BPD can be seen in Appendix B). During the second phase of the design methodology we minimize the number of the elements using some guidance and considering the common average usage rate of BPMN elements. The process depicted as far as here should be **correct**, that is all the elements are represented following the BPMN guidelines, **complete**, representing the process as-is, **compliant** with BPMN specifications.

4.1.2 Phase 2: Logical Modeling

During the second phase of the design methodology we start reasoning about the diagram we created during the first phase. The first phase rules provided some simple steps to perform the first necessary tread to design a business process: to give the textual description of a process a graphical look. Now we present

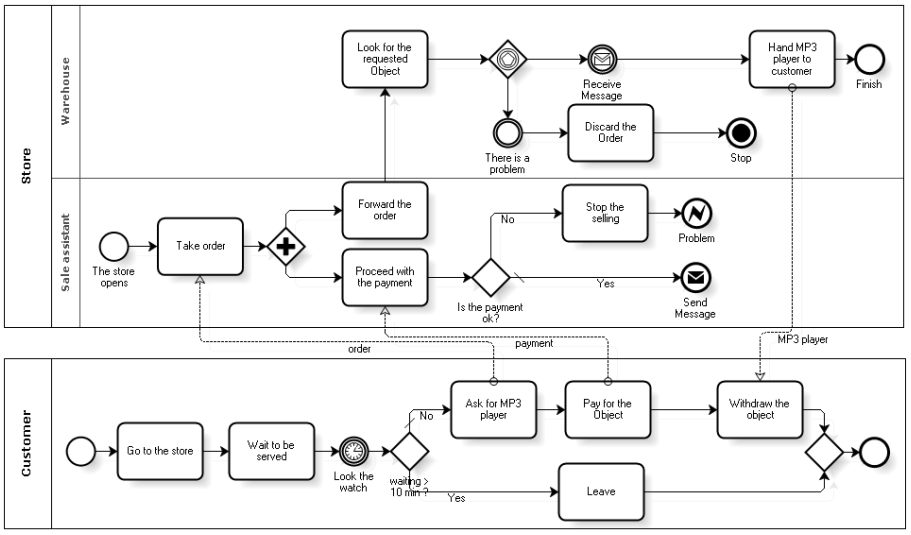


Figure 4.5: Rule 1.5: The first phase output process, ready to be refined

refinement rules to improve the business process diagram. At first, some preliminary assumptions have to be stated. BPMN specifications provide a large variety of symbols, to allow modelers to describe a high variety of processes. However, only a small set of symbols are commonly used. Muehlen and Recker in [68] present the result of a very large survey on the use of BPMN in different contexts (Web, Consulting, Seminar). The three different contexts are quite aligned on the results. They discovered that the six most used BPMN elements are: Normal Flows, Task, End Event, Start Event / Event, Pool, Data-Based XOR Gateway. They call this set the **common core of BPMN**. To the contrary, there are elements that are not considered at all, like Compensation Association, Multiple End Event, Cancel Intermediate Event, Intermediate Exception, Multiple Start Event, Compensation End Event (see Fig. 4.6).

Looking at the number of the elements used for a diagram, they found that none of the diagrams used more than 15 different BPMN constructs, and none used less than 3. The models themselves could contain more elements, but a model with, e.g., 5 Tasks connected by Sequence Flows uses in practice only the Task symbol and the Sequence Flow symbol (only 2 symbols). The average subset of BPMN elements used in these models consisted of 9 different symbols, meaning that the average BPMN model uses less than 20% of the available vocabulary. The diagram we depicted at the end of the first phase of the methodology uses about 15 different elements, and, thus, we can argue that it could be positioned between the most complex processes (in terms of number of BPMN elements used), as it is possible to see in Fig. 4.7.

This analysis leads us to make some assumptions about the process we drawn in the first phase of the methodology. One model to be **relevant** must describe as better as possible the related processes. We have to deal with two different points

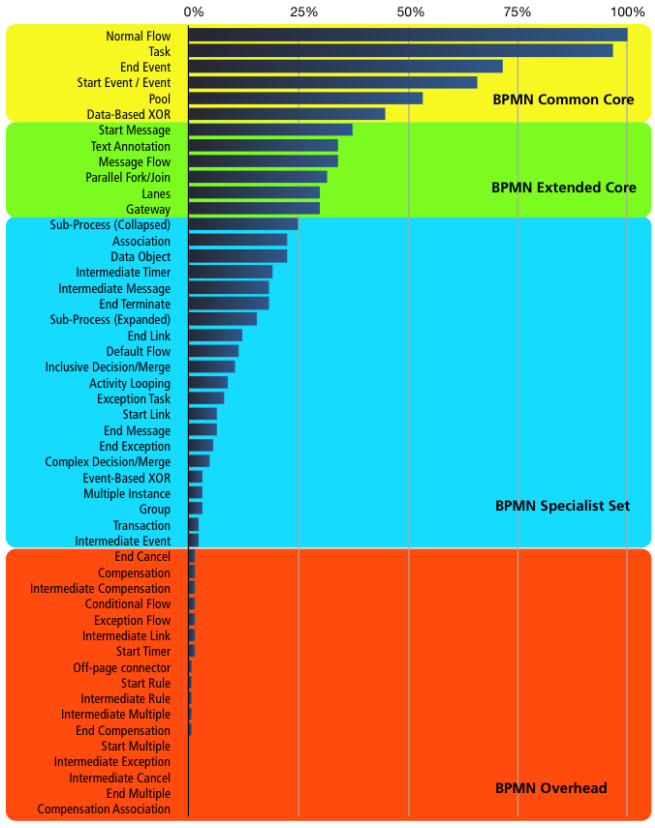


Figure 4.6: The number of BPMN constructs used across which percentage of the collected diagrams

of view. On one hand, we have to discover if we have described all the aspects of the process using the most appropriate elements – BPMN provides a lot of useful objects. On the other hand we must ask ourselves if everything inside the model is relevant. To be more precise, if we could delete some unnecessary elements or represent the same object with less (and more appropriate) elements.

Before proceeding with the definition of the rules, it is important to consider that there are many ways to reduce a business process. Reduction rules have been successfully used either as a stand alone approach [42, 43], or as an engineering approach used to reduce the complexity of the process models [57, 26] to verify correctness of process models. These approaches where applied to other processes modeling representations [3], like EPC or Petri Nets. All these tools could be easily adopted to accomplish the rules of this second phase.

Rule 2.1: Participant dependencies. One actor, one service, one generic entity or role can be identified as a Participant when it does not have functional or hierarchical dependencies with other entities or roles. Looking at the diagram, it has to be considered the opportunity to identify Lanes which could

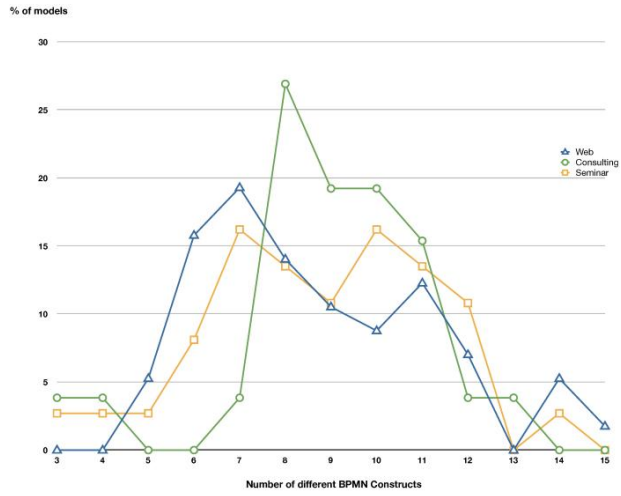


Figure 4.7: The syntactic complexity of the BPMN models in relationships with the percentage of models

be merged together to be turned into a Pool or Pools that could be splitted in more than one Lane to refine the model.

Looking at the example, we have one sale assistant and one warehouseman who are employed to the same store. Thus, we have represented them with two Lanes belonging to one single Pool called "Store". The Customer and the Store are independent each other, so, they are two different Pools. This is a very basic example, but sometimes could be very hard to state if two or more participants have not strict relationships. If we model a company with more than one office, we probably have to design one single Pool splitted in several Lanes. To the contrary, one process representing a patient who ask his doctor for an exam to be taken at the hospital would be modeled with three different Pools.

Rule 2.2: Transform the Activities. Consider each Activity (Task or Sub-Process). Having in mind the intended abstraction level, transform Tasks in Sub-Processes (which correspond to reach an higher level of description) and/or Sub-Processes in Tasks (lower details level). Then, it is possible to consider the following guidance:

- **Rule 2.2.1: Activities ordering.** For each participant, apply the precedence rules to the activities.
- **Rule 2.2.2: Activities types.** For each activity (Task or Sub-Process) apply the right type according to the semantic of the work performed.
- **Rule 2.2.3: Activities substitution.** This is a two-directions rule. Both directions give as result a simplified version of the process. The first direction consists of substitute an activity (namely, in most cases, a Task) with an Event with the same behavior or which can perform

the same activity work: Events have no weight in the process total cost counting. The second direction lies in grouping a set of Tasks in a Sub-Process, making the process more understandable [58]. Moreover, there could happen that the process abstraction level (and, thus, its granularity) changes from the first phase. We could deal with the necessity to replace one Task with one Sub-Process or, on the other way, one Sub-Process with one Task. Make these changes sparingly to avoid an excessive difference between the first phase diagram and the second phase model. Finally, consider if the Rules 2.2.1 and 2.2.2 should be applied again.

We now pass to a detailed description of these rules. During the first phase we have positioned all the activities inside the diagram paying attention especially to assign an activity to the right participant. Now we pass to check activities precedences. Such precedences could be provided explicitly inside the process description text or implicitly, looking at the process semantics. Basing the structuring of one process only on content-dependent rules makes the adoption of a clear and shared taxonomy a very important asset for enterprises. A number of activities without precedence rules is a set, but we want to obtain a sequence (the concept of *sequence* is distinct from that of a *set*, in that the elements of a sequence appear in some order). The strategy we choose to adopt in this case is to create a system of precedence rules (quite similar to the system of linear equations concept) where variables are the process activities. Without any precedence rule, if a process has n_A activities, they could origin $n_A!$ different permutations. Using our example process, consider the Customer process. There are six different activities (in this case they are all Tasks, but this does not effect the strategy). To simplify the example readability we apply a letter from A to F to each activity. So we are in the condition depicted in Fig. 4.8.

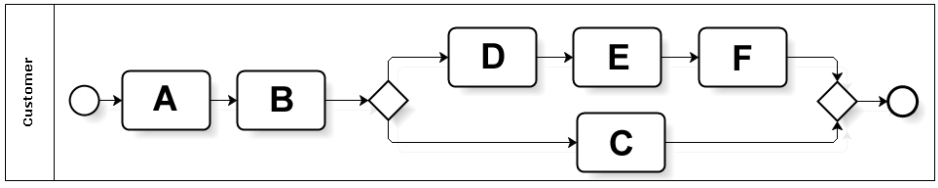


Figure 4.8: Process activities to be inferred

From the text description we can infer the following precedence rules, either implicit or explicit, temporal or functional:

- A must be the first activity
- C and F are the last two activities
- D and C come after B
- C and D,E,F should not be positioned on the same path
- E and F come after D

- B comes after A and before C and D

As for a classical equations system, if we can collect many (useful) rules as the number of the variables we can precisely arrange actions following the process flow. Collecting less than needed rules prevents the possibility to clearly model the process flow. The most suitable goal is to have no degrees of freedom in activities positioning. Another risk is to collect more rules than those requested. If there are equivalent rules, it is possible to simplify the system. Otherwise there could be inconsistencies or errors in the process description or in the first phase model. There are works (as, for example, [35]) which clearly state this problem providing some methodology to accomplish this task.

This procedure can be applied also in the case we have to describe an Ad-Hoc Sub-Process showing also its tasks. Starting from a collapsed Ad-Hoc Sub-Process, expand and transform it in one Sub-Process showing its internal Tasks. Than apply the procedure tracing the Sequence Flows to connect the activities. At last add, if needed, all the other elements like Events or Gateways. The resulting Sub-Process will become no more an Ad-Hoc Sub-Process but an Embedded Sub-Process holding a complete business process.

Let us discuss now the other two rules. To apply the correct type to Tasks and Sub-Processes should be an easy step. BPMN specifications provide different activities types especially for Tasks. There could be present inside the model Service, User, Send, Receive, Script, Manual and Reference Tasks, depending on the Tasks behavior. Every type extends the common attribute set for activities adding particular features. Be careful to consider also general loop, compensation and multi-instance types, as provided by BPMN specifications for both Tasks and Sub-Processes. Assigning the correct type will increase the readability of the diagram making it easier to be understood by different persons and shared between offices, services, companies.

Finally, consider the two directions defined above for the third rule. The use of subprocesses 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, the lack of precise guidelines turns out to be a major impediment for applying modularity in a systematic way. A systematic and grounded approach to find the optimal modularization of a process model is still missing [39]. Therefore, looking at the process detect sections which could be replaced by Sub-Processes. This makes the process more understandable [58]. Following the opposite direction, instead, analyze the process looking for all those activities (in this case, they should be Tasks) which could be replaced by Events. For example, if the aim of a Task is only to wait for an incoming message, the Task can be replaced by an Intermediate Event of type Message. In our running example the "Send Message" Activity can be modeled using an Event rather than a Task.

Rule 2.3: Events. Perform some checks upon Events, having in mind the goal to build a relevant and economical model, easily readable and understandable. BPMN specifications provide some guidelines about when and how Events should be used.

A Start Event is optional, but if there is an End Event in the Process, then one Start Event must be explicitly defined. To the contrary, End Events are optional,

but if there is a Start Event, then there must be at least one End Event. It is reasonable to choose a modeling style opting for to use Start and End Events if they are meaningful for the process. In fact, if a Start (End) Event is not used, it does not have any trigger (result). There may be multiple Start (End) Events within a single level of a process, but the behavior of the process may be harder to understand if there are multiple Start (End) Events. It is recommended that this feature be used sparingly and that the modeler be aware that other readers of the diagram may have difficulty understanding the intent of the diagram. But, while reading a process with one Start Event and more than one End Event is a common scenario (very often a process has one starting condition but may have multiple ends depending on the complexity of the process – the number of parallel or conditional paths that may be followed), dealing with a process which may start in several ways is disorienting. In [12], Mendling and Decker considered six different process notations and syntax: open workflow nets, YAWL, event-driven process chains, BPEL (the code, not a graphical representation), UML activity diagrams, and BPMN. They determined how an entry point is represented in each of these notations: start places (such as in open workflow nets), start events (such as in BPMN) and start conditions (such as in event-driven process chains). Having multiple start events in a process causes all sorts of problems in terms of understandability and soundness, and, endorsing the BPMN specifications suggestion, they do not recommend this in general; however, since the notations support it and therefore it can be done in practice, it is reasonable to use multiple Start Events only if the process is complex and/or the starting conditions are not obvious.

Rule 2.4: Gateways. Identify all those Gateways that are not relevant for the model and replace them with implicit Gateways or with Events attached to the Activities boundaries. Check also if each splitting Gateway has its corresponding merge Gateway, even if it is implicitly defined.

Look at the process diagram in Fig. 4.9, which represents the result process at the end of the Rule 2.3. The emphasized Gateways can be replaced by implicit ones or by intermediate events attached to activities boundaries. BPMN specifications provide some rule to express implicit splitting, branching, joining and merging paths. If multiple paths exit from an activity or from an event, they have to be considered as they were the output paths of a parallel (AND) gateway, meaning that all the paths will be executed at the same time. If one activity has conditional outgoing Sequence Flows, instead of common uncontrolled Sequence Flows, the branching point is represented by an Inclusive Gateway. To replace explicit elements with their implicit form is very useful if the diagram of the process has too much symbols, making it not so easily understandable.

Sometimes a Gateway could be changed with an Intermediate Event. It depends quite exclusively from the semantics and the behavior of the Gateway. Very often in these cases Gateway are used to model choices which generate exception flows. In the example, the Customer process has one Gateway which can be modeled attaching a Timer Intermediate Event to the boundary of the previous activity. Also in the Warehouseman work flow the selected Gateway can be substituted by an Intermediate Event placed on the boundary of the previous Task

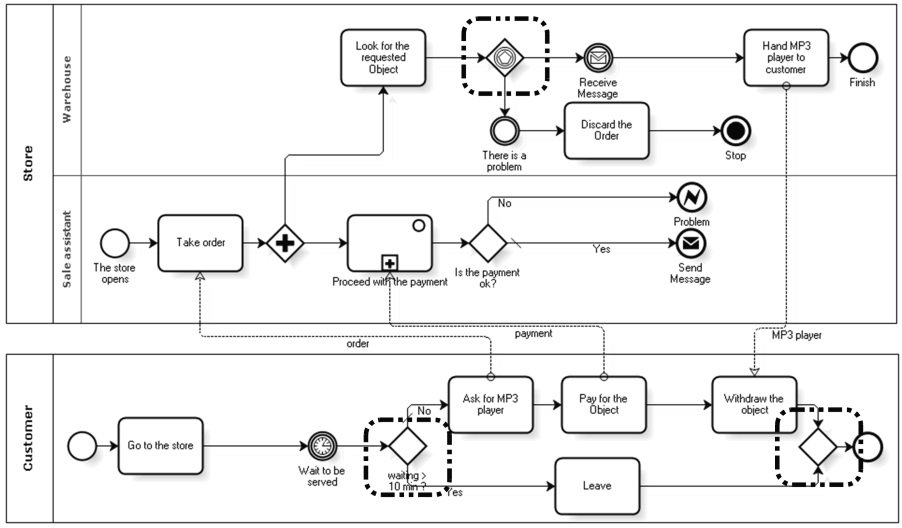


Figure 4.9: Rule 2.4: The example process at the end of Rule 2.3 with some to be deleted Gateways enphasized

and generating an exception flow.

Rule 2.5: Patterns analysis. Once all the elements of the model have been refined, we must consider the flows. Reduce where possible the flows to well known patterns.

The most spread work on patterns is the workflow patterns definition by van der Aalst [55], used also to develop the flow patterns for BPMN. It is not feasible to change a pattern at the end of the second phase. This rule is positioned here because during the application of the other rules the diagram configuration could have been changed. Applying patterns from the beginning is a very useful way to model flows. Another help come from some really new works (among the other, [17] is remarkable) on applying pattern during modeling or to already modeled processes. Although the business process community has put a major emphasis on patterns only limited support for using patterns, in today's business process modeling tools can be found. There is a lack of support for the users to correctly applying these patterns to many incorrectly modeled business processes. We decided to integrate in this rule this topic to give our contribution with this design methodology. In their work, Gschwind, Koehler and Wong describe an extension of a business process modeling tool with patterns. They distinguish three scenarios of pattern application: refinement of a single control-flow edge by applying a block-oriented pattern compound, application of a pattern compound to a pair of selected control-flow edges, application of a basic pattern to a set of selected control-flow edges. The result of their work is an extension which supports users to apply patterns to incorrectly modeled diagrams. While it is still better to apply patterns during the first phase, using some strategies or guidance like the

one developed by Gschwind, Koehler and Wong it is still possible to correct old process designs or diagrams incorrectly modeled during the first phase, as well as to add/remove elements. We remark how it is important this rule after the model is changed because of the effects of the other second phase rules execution.

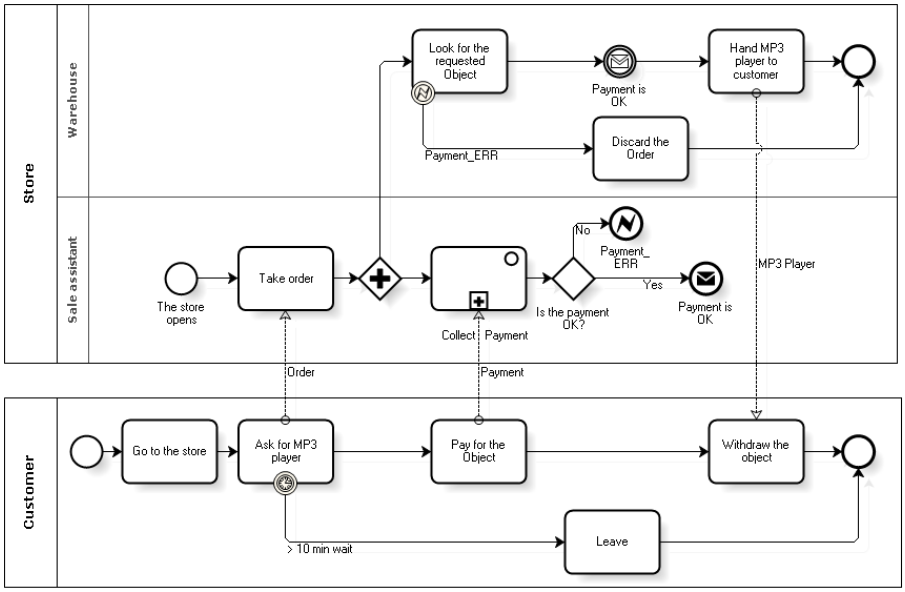


Figure 4.10: Rule 2.5: The final version of the example process, at the end of the Phase 2

Second Phase Concluding Remarks. At the end of this phase (the final version of the example process is depicted in Fig. 4.10 and also in Appendix B) the process should be still correct, complete and compliant as at the end of the first phase. Moreover, it should be **relevant** (all the symbols used are meaningful and appropriate) and **economical** (there are not useless elements). If all the rules have been correctly applied, the resulting model will be more easily readable and understandable.

The process now is ready to be translated in a suitable way, such as our BPeX XML serialization schema.

4.1.3 Phase 3: Physical Modeling

During the last phase, the process should not be further modified. The goal of this phase is to turn the second phase output model into a physical format. We choose to use our BPeX XML-Schema because of its natural correspondence with the BPMN model. To accomplish this task we use a mix of top-down and bottom-up methodologies. We will use the top-down methodology to derive the XML-tree structure starting from the model diagram, while we apply the bottom-up method-

ology to enrich the elements definitions with all the constructs and attributes. The XML serialization of the example process can be viewed in Appendix B.

It is important to notice how linear is the execution of this third phase if our BPeX serialization is used. Having a natural correspondence with the BPMN diagrams structure, it is possible to have the XML serialization of a process building it from scratch without the need of a software tool. BPeX serialization gives modeler the full control over the code. It has a simpler structure and this makes it more easily readable and understandable than the BPMN 2.0 new purposes. As we have already discussed more in details in Chap. 3, being a native BPMN serialization built from scratch, it has not the disadvantages or weaknesses of other purposes like XPDL or BPEL.

Rule 3.1: BPD, Swimlanes and Processes. We start the top-down serialization creating an empty Business Process Diagram, adding all the Participants and the Processes.

The XML-Schema structure follows the BPMN definition and, thus, if one Pool holds only one Lane, the Lane has to be defined sharing the same name of the Pool. Processes and Participants have not a graphical look: they are defined within their connected Pool, according to the BPeX model architecture. Listing 4.1 shows the BPeX XML code for the running example.

```
<?xml version="1.0" encoding="UTF-8"?>                                          1
<BPD bpex:Id="BPD_001" bpex:Name="Customer/Store␣Example">                       2
  <Pool bpex:ParticipantRef="Part_01" bpex:Name="Store" bpex:Id="Pool_001">      3
    <Process bpex:Name="Store␣Process" bpex:Id="Proc_01"/>                       4
    <Participant bpex:Id="Part_01"/>                                            5
    <Lane bpex:Name="Warehouseman" bpex:Id="Lane_001"/>                         6
    <Lane bpex:Name="Sale␣Assistant" bpex:Id="Lane_002"/>                       7
  </Pool>                                                                       8
  <Pool bpex:ParticipantRef="Part_02" bpex:Name="Customer" bpex:Id="Pool_002">9
    <Process bpex:Name="Customer␣Process" bpex:Id="Proc_02"/>                   10
    <Participant bpex:Id="Part_02"/>                                            11
    <Lane bpex:Name="Customer" bpex:Id="Lane_003"/>                             12
  </Pool>                                                                       13
</BPD>                                                                          14
```

Listing 4.1: Rule 3.1 Definition of BPD/Pool/Lane elements

Rule 3.2: Activities. For each Lane, add every Activities, starting from the beginning of the Process through the end.

We suggest to start with the Tasks definition and then with the Sub-Process description. While Tasks are independent activities, Sub-Process may refer to other Processes (Reference Sub-Processes) which must be early defined, or to other Sub-Processes already declared in the Diagram. Listing 4.2 shows an excerpt for Tasks and Sub-Processes definition.

To write the Activities definitions at first, permits users to more easily manage the high number of Sequence Flows definitions. During the execution of the following Rules, it will be possible to pinpoint Sequence Flows a little at a time, bringing every element definition together with its related Sequence Flows statements.

```
<?xml version="1.0" encoding="UTF-8"?>
<BPD bpex:Id="BPD_001" bpex:Name="Customer/Store␣Example">
  <Pool bpex:ParticipantRef="Part_01" bpex:Name="Store" bpex:Id="Pool_001">
    <Process bpex:Name="Store␣Process" bpex:Id="Proc_01"/>
    <Participant bpex:Id="Part_01"/>
    <Lane bpex:Name="Warehouseman" bpex:Id="Lane_001">
      <Task bpex:ActivityType="Task"
        bpex:Name="Look␣for␣the␣requested␣Object" bpex:Id="T_006"/>
      <Task bpex:ActivityType="Task"
        bpex:Name="Discard␣the␣Order" bpex:Id="T_007"/>
      <Task bpex:ActivityType="Task"
        bpex:Name="Hand␣MP3␣player␣to␣the␣customer" bpex:Id="T_008"/>
    </Lane>
    <Lane bpex:Name="Sale␣Assistant" bpex:Id="Lane_002">
      <Task bpex:ActivityType="Task"
        bpex:Name="Take␣order" bpex:Id="T_009"/>
      <Sub-Process bpex:SubProcessType="Embedded"
        bpex:ActivityType="Sub-Process" bpex:Name="Collect␣Payment"
        bpex:Id="SP_001"/>
    </Lane>
  </Pool>
  <Pool bpex:ParticipantRef="Part_02" bpex:Name="Customer" bpex:Id="Pool_002">
    <Process bpex:Name="Customer␣Process" bpex:Id="Proc_02"/>
    <Participant bpex:Id="Part_02"/>
    <Lane bpex:Name="Customer" bpex:Id="Lane_003">
      <Task bpex:ActivityType="Task" bpex:Name="Go␣to␣the␣Store"
        bpex:Id="T_001"/>
      <Task bpex:ActivityType="Task" bpex:Name="Ask␣for␣MP3␣Player"
        bpex:Id="T_002"/>
      <Task bpex:ActivityType="Task" bpex:Name="Leave"
        bpex:Id="T_003"/>
      <Task bpex:ActivityType="Task" bpex:Name="Pay␣for␣the␣Object"
        bpex:Id="T_004"/>
      <Task bpex:ActivityType="Task" bpex:Name="Withdraw␣the␣object"
        bpex:Id="T_005"/>
    </Lane>
  </Pool>
</BPD>
```

Listing 4.2: Rule 3.2: Definitions of Tasks and Sub-Processes

Rule 3.3: Events. For each Lane define all the Events, starting from the Start Events and finishing with the End Events. For each Event, add immediately also the definition of all in- and outgoing Sequence Flows which connect Events with Activities.

In the List. 4.3 an example definition for some Events and related Sequence Flows is shown.

Rule 3.4: Gateways. Similarly to the previous Rules, add to each Lane the Gateways definitions. For each Gateway, add immediately also the definition of in- and outgoing Sequence Flows which connect Gateways with Activities and/or Events.

Just one brief note to this Rule: while adding the Gateways related Sequence Flows, it is important to mark the right outgoing Sequence Flows as Default (whether it is present).

Rule 3.5: Flows. Flows are a bit more complicated. First of all it is required to distinct between the Message Flows definitions and Sequence Flows def-

```
<?xml version="1.0" encoding="UTF-8"?>
<BPD bpex:Id="BPD_001" bpex:Name="Customer/Store␣Example">
  <Pool bpex:ParticipantRef="Part_01" bpex:Name="Store" bpex:Id="Pool_001">
    <Process bpex:Name="Store␣Process" bpex:Id="Proc_01"/>
    <Participant bpex:Id="Part_01"/>
    <Lane bpex:Name="Warehouseman" bpex:Id="Lane_001">
      <IntermediateEvent bpex:EventType="Intermediate"
        bpex:Name="Payment␣is␣OK" bpex:Id="IE_001">
        <Trigger bpex:EventDetailType="Message" bpex:Id="Tr_001"/>
      </IntermediateEvent>
      <IntermediateEvent bpex:EventType="Intermediate"
        bpex:Name="Payment_ERR" bpex:Id="IE_002">
        <Trigger bpex:EventDetailType="Error" bpex:Id="Tr_002"/>
        <Target>T_006</Target>
      </IntermediateEvent>
      <EndEvent bpex:EventType="End" bpex:Name="" bpex:Id="EE_001">
        <Result bpex:EventDetailType="None" bpex:Id="Res_001"/>
      </EndEvent>
    </Lane>
    <Lane bpex:Name="Sale␣Assistant" bpex:Id="Lane_002">
      <StartEvent bpex:EventType="Start" bpex:Name="The␣store␣opens"
        bpex:Id="SE_001">
        <Trigger bpex:EventDetailType="None" bpex:Id="Tr_003"/>
      </StartEvent>
      <EndEvent bpex:EventType="End" bpex:Name="Payment_ERR"
        bpex:Id="EE_002">
        <Result bpex:EventDetailType="Error" bpex:Id="Res_002"/>
      </EndEvent>
      <EndEvent bpex:EventType="End" bpex:Name="Payment␣is␣OK"
        bpex:Id="EE_003">
        <Result bpex:EventDetailType="Message" bpex:Id="Res_003"/>
      </EndEvent>
    </Lane>
  </Pool>
  <Pool bpex:ParticipantRef="Part_02" bpex:Name="Customer" bpex:Id="Pool_002">
    <Process bpex:Name="Customer␣Process" bpex:Id="Proc_02"/>
    <Participant bpex:Id="Part_02"/>
    <Lane bpex:Name="Customer" bpex:Id="Lane_003">
      <StartEvent bpex:EventType="Start" bpex:Name="" bpex:Id="SE_002">
        <Trigger bpex:EventDetailType="None" bpex:Id="Tr_004"/>
      </StartEvent>
      <IntermediateEvent bpex:EventType="Intermediate"
        bpex:Name=">␣10␣min" bpex:Id="IE_003">
        <Trigger bpex:EventDetailType="Timer" bpex:Id="Tr_005"/>
        <Target>T_002</Target>
      </IntermediateEvent>
      <EndEvent bpex:EventType="End" bpex:Name="" bpex:Id="EE_004">
        <Result bpex:EventDetailType="None" bpex:Id="Res_004"/>
      </EndEvent>
    </Lane>
  </Pool>
</BPD>
```

Listing 4.3: Rule 3.3: Events definitions

initions. Message Flows must be defined as BPD children, while Sequence Flows as Pool children.

If all the previous Rules have been followed, most of Sequence Flows should have been already defined. Anyhow, start now with the Message Flows definition. They are very often less than Sequence Flows and their properties are easier to be set. Then, for each Pool, add all the remaining Sequence Flows, checking also the attributes of the previous defined ones. Start from the beginning of the process and follow every path. The concept of Token introduced by BPMN specifications

```
<?xml version="1.0" encoding="UTF-8"?>
<BPD bpex:Id="BPD_001" bpex:Name="Customer/Store␣Example">
  <Pool bpex:ParticipantRef="Part_01" bpex:Name="Store" bpex:Id="Pool_001">
    <Process bpex:Name="Store␣Process" bpex:Id="Proc_01"/>
    <Participant bpex:Id="Part_01"/>
    <Lane bpex:Name="Warehouseman" bpex:Id="Lane_001">...</Lane>
    <Lane bpex:Name="Sale␣Assistant" bpex:Id="Lane_002">
      <Gateway bpex:Name="" bpex:Id="GW_001" bpex:GatewayType="Parallel">
        <Gates bpex:OutgoingSequenceFlowRef="SF_001"/>
        <Gates bpex:OutgoingSequenceFlowRef="SF_002"/>
      </Gateway>
      <Gateway bpex:Name="Is␣the␣payment␣OK?" bpex:Id="GW_002"
        bpex:GatewayType="Data-Based␣Exclusive">
        <Gates bpex:OutgoingSequenceFlowRef="SF_003"/>
        <Gates bpex:OutgoingSequenceFlowRef="SF_004"/>
      </Gateway>
    </Lane>
  </Pool>
  <Pool bpex:ParticipantRef="Part_02" bpex:Name="Customer" bpex:Id="Pool_002">
    <Process bpex:Name="Customer␣Process" bpex:Id="Proc_02"/>
    <Participant bpex:Id="Part_02"/>
    <Lane bpex:Name="Customer" bpex:Id="Lane_003">...</Lane>
  </Pool>
</BPD>
```

Listing 4.4: Rule 3.4: The example process with Gateways defined

may be used to ease this task. Listing 4.5 shows how Message Flows and Sequence Flows could be defined using BPeX serialization.

Now it is possible to add to the XML tree all the other attributes and features, following a bottom-up methodology direction.

Third Phase Concluding Remarks. The BPeX XML-Schema aids users not to omit something integrating a full referential integrity support. BPeX XML-Schema uses XPath functions like the `xs:assert` and `xs:alternative` to restrain users not to miss important constraints. The XML instance document will be right validated only if all the mandatory attributes will have been declared, all the references will have been satisfied and all the definitions will be compliant with the BPMN requirements.

4.1.4 Business Process Normal Form

We introduce a lightweight concept of *Normal Form* applied to a business process, meaning that the processes modeled following these three-phases sets of rules will have some characteristics which guarantee some basic properties (e.g., the process diagram is compliant with BPMN specifications, the business process diagram can model correctly the behavior of the business process, and so forth). We can define *Business Process Normal Form* (or BPNF) as the desired form a business process model (with 'model' we mean both the graphical representation and its serialization together) should have.

If the three-phases methodology we presented in this chapter is applyied the modeled business process will be in BPNF. One business process model is said to be in BPNF if it complies with all of the requirements stated in the three phases of the design methodology. The more important requirements are resumed in

Tab. 4.5. If one business process model satisfies the minimum requirements it should also be easily readable and understandable.

	Correct, Complete, Compliant	Relevant	Economical
Phase 1	•		
Phase 2	•	•	•
Phase 3	•	•	•

Table 4.5: Summary of minimal phases requirements

Inside either the two main purposes for BPMN 2.0 specifications there is the definition of BPMN Complete Conformance or Full Compliance. We spend no space to describe these definitions because they are not yet approved and they may change. But it is interesting to notice how recently companies and organizations start to point their attention also on this topic. Their definition of BPMN Complete Conformance or Full Compliance does not overlap our definition of BPNF, because the aims are different. While we give a definition of the desired form for a business process model, the BPMN Complete Conformance / Full Compliance given for BPMN 2.0 describes what characteristics a software must have to claim BPMN 2.0 compliance or conformance.

4.1.5 Conclusions

The three-phase methodology needs a weigh tests stage to fulfill all the missing particular cases. Anyhow, the aim of the methodology is to provide a set of well-organized design steps to guide users represent their business processes in a form which owns some important properties and features. This is a concrete, completely new, first real answer to the lack of a design methodology for business processes which makes a massive use of BPMN and emerging related specifications. The example process used to illustrate the methodology is not complete and does not model all the process events should occur, but it was chosen to represent as better as possible the greater number of the most commons constructs used in business process modeling.

```
<?xml version="1.0" encoding="UTF-8"?>
<BPD bpex:Id="BPD_001" bpex:Name="Customer/Store␣Example">
  <Pool bpex:ParticipantRef="Part_01" bpex:Name="Store" bpex:Id="Pool_001">
    <Process bpex:Name="Store␣Process" bpex:Id="Proc_01"/>
    <Participant bpex:Id="Part_01"/>
    <Lane bpex:Name="Warehouseman" bpex:Id="Lane_001">...</Lane>
    <Lane bpex:Name="Sale␣Assistant" bpex:Id="Lane_002">...</Lane>
    <SequenceFlow bpex:Name="" bpex:SourceRef="GW_001"
      bpex:TargetRef="T_006" bpex:Id="SF_001"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="GW_001"
      bpex:TargetRef="SP_001" bpex:Id="SF_002"/>
    <SequenceFlow bpex:Name="No" bpex:SourceRef="GW_002"
      bpex:TargetRef="EE_002" bpex:Id="SF_003"/>
    <SequenceFlow bpex:Name="Yes" bpex:SourceRef="GW_002"
      bpex:TargetRef="EE_003" bpex:Id="SF_004" bpex:ConditionType="Default"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="SE_001"
      bpex:TargetRef="T_009" bpex:Id="SF_005"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_009"
      bpex:TargetRef="GW_001" bpex:Id="SF_005"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="SP_001"
      bpex:TargetRef="GW_002" bpex:Id="SF_006"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_006"
      bpex:TargetRef="IE_001" bpex:Id="SF_007"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="IE_001"
      bpex:TargetRef="T_008" bpex:Id="SF_008"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_008"
      bpex:TargetRef="EE_001" bpex:Id="SF_009"/>
    <SequenceFlow bpex:Name="Payment_ERR" bpex:SourceRef="IE_002"
      bpex:TargetRef="T_007" bpex:Id="SF_010"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_007"
      bpex:TargetRef="EE_001" bpex:Id="SF_011"/>
  </Pool>
  <Pool bpex:ParticipantRef="Part_02" bpex:Name="Customer" bpex:Id="Pool_002">
    <Process bpex:Name="Customer␣Process" bpex:Id="Proc_02"/>
    <Participant bpex:Id="Part_02"/>
    <Lane bpex:Name="Customer" bpex:Id="Lane_003">...</Lane>
    <SequenceFlow bpex:Name="" bpex:SourceRef="SE_002"
      bpex:TargetRef="T_001" bpex:Id="SF_012"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_001"
      bpex:TargetRef="T_002" bpex:Id="SF_013"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_002"
      bpex:TargetRef="T_004" bpex:Id="SF_014"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_004"
      bpex:TargetRef="T_005" bpex:Id="SF_015"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_005"
      bpex:TargetRef="EE_004" bpex:Id="SF_016"/>
    <SequenceFlow bpex:Name="␣10␣min" bpex:SourceRef="IE_003"
      bpex:TargetRef="T_003" bpex:Id="SF_017"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_003"
      bpex:TargetRef="EE_004" bpex:Id="SF_018"/>
  </Pool>
  <MessageFlow bpex:Name="Order" bpex:SourceRef="T_002"
    bpex:TargetRef="T_009" bpex:Id="MF_001">
    <Message bpex:Name="Order" bpex:FromRef="Part_02"
      bpex:ToRef="Part_01" bpex:Id="MSG_001"/>
  </MessageFlow>
  <MessageFlow bpex:Name="Payment" bpex:SourceRef="T_004"
    bpex:TargetRef="SP_001" bpex:Id="MF_002">
    <Message bpex:Name="Payment" bpex:FromRef="Part_02"
      bpex:ToRef="Part_01" bpex:Id="MSG_002"/>
  </MessageFlow>
  <MessageFlow bpex:Name="MP3␣Player" bpex:SourceRef="T_008"
    bpex:TargetRef="T_005" bpex:Id="MF_003">
    <Message bpex:Name="MP3␣Player" bpex:FromRef="Part_01"
      bpex:ToRef="Part_02" bpex:Id="MSG_003"/>
  </MessageFlow>
</BPD>
```

Listing 4.5: Rule 3.5: All Sequence Flows and Message Flows are defined

Chapter 5

Business Process Views

> Look wide! And when you think you are looking wide, look wider still.
>
> *Sir Robert S. Smyth Lord Baden-Powell*

BPMN 2.0 RFP claims for an "enhanced support for differing perspectives (i.e., alternative displays) of a model to address different modeling and analysis interests". Effectively, to deal with complex business processes diagrams, with plenty of elements, is not an easy work. Now that BPMN is gaining even more interest from companies and the number of implementations is rising up, new needs are coming out. Among others, there are two precise expectations: to have a methodology to accomplish common tasks – and in particular to aid users to model processes – and to be able to differentiate how processes have to appear to different users with different roles.

Having already presented our proposal for a business process design methodology in the previous chapter, now we introduce a new feature which can help companies to model different perspectives of their business processes, allowing also to add a users management support (at this moment this is only a rough draft feature which could be investigated more in details once the new BPMN 2.0 standard will be published): *Business Process Diagram Views* (BPDV). Views could also be useful to enable new features like information hiding or integration with users privileges. Views provide a way to make complex and commonly issued queries easier to compose. Views allow obsolete processes, and programs that reference them, to survive reorganization. Moreover, they add a security aspect to allow different users to see the same data in different ways. In particular, it is a useful feature that not everyone should have access to every piece of information.

The concept of views is borrowed from the database terminology. There are different kinds of BPDVs, depending on the purpose for which a view is requested. We could have views representing a query result or pointing out one process among others as well as highlighting some process properties.

5.1 Business Process Diagram Views

A Business Process Diagram View (or simply a *view*) is a diagram that results from a query, but it has its own name and can be treated in many ways (depending on its type as explained below) as if it was a base diagram. Thus a view is a logical *window* on the data and the structure of the base BPD that can be even updated in certain, carefully selected cases.

The concept of *view* could also be used to model the needed different granularity levels of a business process. For instance, one business process with all its Sub-Processes collapsed is a different view(-point) with respect to have full access also to the contained processes description.

One BPDV is not a snapshot of the data nor of the process state at the time a view is created, but it only stores definitions that must be interpreted each time a view is generated. One BPDV is always defined as a subset of Business Process Diagram elements. There are two possible ways to derive one BPDV from one starting BPD: by graphically selecting elements or by executing queries upon the BPD serialization. In both cases we use the term *query* to point out the process by means of which a view can be generated starting from a BPD. As a matter of fact, assuming to have a complete and compliant BPMN serialization in an XML structured format (like, e.g., our BPeX model proposal), we can represent either the graphical elements selection and the query execution through XPath/XQuery statements.

BPDV are more complicated to be defined than relational databases views, especially because business processes views could be generated both on the graphical layout of the process and on its data saving format, while databases views can be executed only on the data saving format (tables). Moreover, databases are very often created by computer scientists, technicians, software engineers and used by business men as they are provided. To take the greatest advantages from business processes modeling, business users ask for a very handy tool. Business process views can satisfy this requirement only if they allow business users to extract from a process the information they really require. Thus, we need to introduce in Sect. 5.2 a taxonomy to clearly classify BPDVs.

There are some works regarding a query language for BPMN, as in [3] with BPMN-Q, but they are at an early stage of formalization. However, there is not at this time an official and complete XML serialization of BPMN. Thus, to show some examples of views we will use our metamodel and its BPeX XML serialization to be able to express queries using XQuery and XPath statements.

5.2 Definition and Classification of Views

Dealing with complex business processes, we can encounter different necessities about tools for representing different aspects of a process. Views can be used to easily accomplish this work.

At present, there is not a strict relationship between the BPMN model graphical representation and an XML serialization. BPEL and XPDL provide two different mappings for BPMN but neither of them is BPMN fully compliant. In fact, OMG calls with its last RFP for a BPMN specification which also must provide an XMI/XML native mapping for BPMN conceptual model and elements, which could express also the BPMN execution semantics. So, we have to deal with two different layers for business process diagram representation: a graphical look and an XML serialization as much suitable as possible. Defining a BPDV as a whatever subset of BPD elements we run against an intrinsic thoughtness. Business users need to manipulate data and processes more freely than a computer scientist is used to do[1]. Elements could be grouped or selected in many ways according to the visualization need of the user (typically, a business man) and each selection or group represents one different BPDV. So, it is useful to introduce a taxonomy to refine and classify all the possible elements sets. The categorization we provide considers several aspects related to *how* one or more elements could be chosen and *which* elements are chosen to become part of a BPDV.

Firstly it is possible to distinguish between views generated by selecting single elements (*extensional* views) or through the execution of some interrogations (*intensional* views). Secondly, queries can be classified depending on which level they are executed in, whether in the graphical representation of the process (*model level* views) or in its XML serialization (*physical level* views). Again, the third classification looks at how many processes are involved in the query, that is if elements come from one process (*intraprocess* views) or from several processes (*interprocess* views.) The last classification refers to the resulting elements set (the view content) and concerns the compliance of the view with a list of properties – in particular if the view could be considered semantically compliant or, even, a new smaller executable process.

We introduce views in business processes also because we aim to use them to make some reasonings, about discuss whether we could update a view, or enriching business processes with users permissions management capabilities. The views classification we introduce in this section will help us to better define these topics later on in this chapter.

5.2.1 Specification-based Classification of Views

As we introduced earlier, we have to deal with two different kinds of models, a graphical layout and an XML representation of business processes. So, there are two different ways to derive a view from a business process model: one is the *intensional* specification of elements and the other is the *extensional* elements specification.

[1] `http://kswenson.wordpress.com/2008/11/25/bpm-is-not-software-engineering/`

Extensional views are those ones which are determined specifying single elements from a business process diagram. Intensional views are the result of the application of a set of rules specifying all the properties required by diagram elements to be considered as part of the view.

Extensional views better apply to graphical selection of diagrams elements, while intensional views better apply to queries executions.

Starting from this preliminary distinction, we can now introduce the other classification types for views.

5.2.2 Level-based Classification of Views

First of all, views can be defined as results of a two-level application: the *model* level and the *physical* level.

Model Level. The definition of *model* is to be interpreted as *model/viewpoint* definition inside the UML specifications: "A model is a description of a physical system with a certain purpose, such as to describe logical or behavioral aspects of the physical system to a certain category of readers. A model is an abstraction of a physical system. It specifies the physical system from a certain vantage point (or **viewpoint**) for a certain category of stakeholders (e.g., designers, users, or customers of the system) and at a certain level of abstraction, both given by the purpose of the model". This kind of views are statically predetermined, with no update capabilities. They are read-only views. It is possible to use the BPMN Artifact element **Group** to show this kind of view. Perhaps, it could be useful to extend the Group notation adding the possibility to draw multiple Group elements sharing the same name to locate scattered elements. There are some different ways to derive a Model Level view, but BPMN-Q [2] is definitely one of the most bright proposal.

Physical Level. This kind of views is defined as the result of queries executions upon the business process serialized form. Forasmuch as the most serialization formats purposes for BPMN provide an XML mapping, we can reasonably state that these views can be defined as results of XML trees handling, or, to be more precise, as results of XPath/XQuery interrogations or XSLT transformations.

In practice, they both could be derived as XPath/XQuery result on the XML instance serialization of one diagram. In particular, using an XML serialization which has a bijective correspondence with BPMN and could represent the whole BPMN elements and attributes set (like BPeX), to define a new view at Model Level corresponds to query the XML serialization of the process to extract the desired elements subset. However, it is important to understand at which level a user needs to define a view. Moreover, if a business process diagram has been serialized using another serialization format, not BPMN fully compliant (e.g., BPEL or XPDL), there could not be the possibility to define the Model Level views simply as queries on the corresponding XML mapping. So, in this section, we will use our BPeX model as reference model to give also some examples.

5.2.3 Direction-based Classification of Views

It is possible to set two different directions of a view, depending on how many processes are involved in the information retrieval process: *interprocess* and *intraprocess* views.

Interprocess Views. Interprocess views are views which extract from a complex diagram (which could be made of multiple processes) a portion included between two key elements, also if the elements belong to different Pools or Lanes. An example of interprocess view can be seen in Fig. 5.1. These views follow the interprocess direction meaning that they take out from different processes a subset of elements.

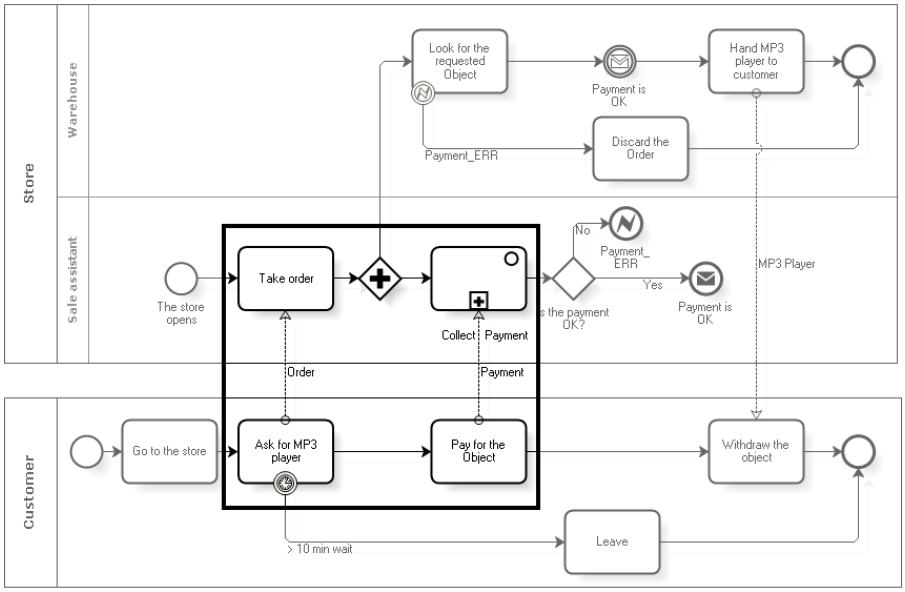
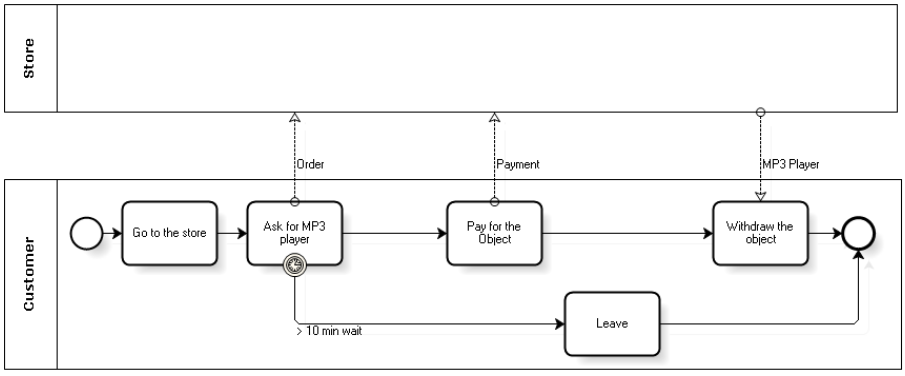


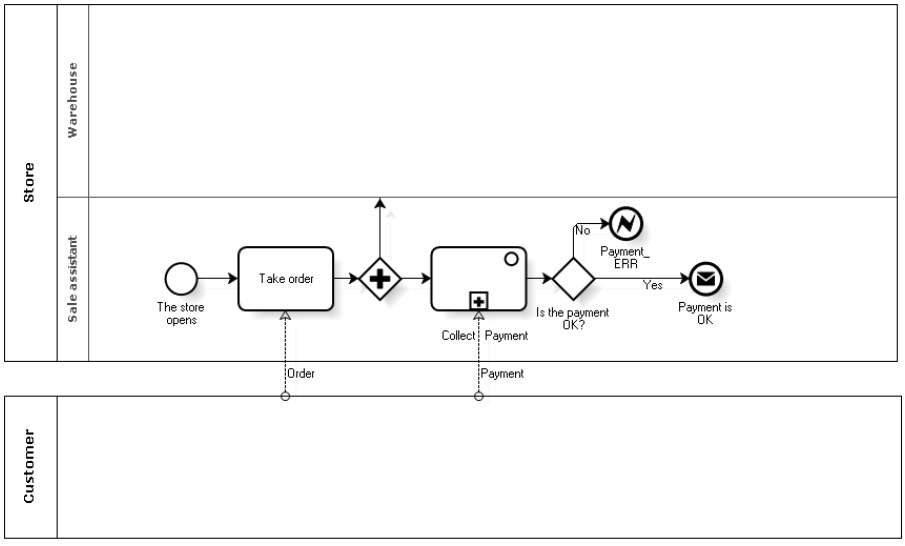
Figure 5.1: An interprocess view on the example process used in Chapter 4

Intraprocess Views. Intraprocess views are views which highlight elements belonging to one Pool or one Lane at a time (or eventually one entire Lane or Pool). Other Pools eventually present in the Process will be omitted or treated as they were black Pools. They highlight one process and its relationships with the others. One example is to consider BPMN processes types classification (Private, Abstract, Collaboration processes). Private processes are an example of the application of this view direction, in which only the internal (private) elements of one single Pool are shown. Global processes are another example in which the view corresponds to the whole diagram. But also the union of Private and Abstract process types could be intended as an example of an intraprocess view in which one process is completely disclosed with also all the relationships with the other

related processes depicted as we were modeling an Abstract process. Figure 5.2(a) shows an example of an intraprocess view executed on different Pools, showing one complete (private) process with the messages exchange involving another process (abstract) clearly specified.



(a) Intraprocess view on Pools



(b) Intraprocess view on Lanes

Figure 5.2: Two examples of Intraprocess views

This kind of view is not provided natively by BPMN specification, even if it could be considered useful in many cases. This concept to use intraprocess views to broaden the representation opportunities of a business process diagram can be easily extended also to Lanes. Imagine to have the need to highlight with a view one single Lane rather than an entire Pool. Lanes have been introduced in BPMN only to better organize the elements inside a Process. They are not connected

each other through Message Flows, and, besides, we can not ignore Sequence Flows because the behavior of the process could notably change. Thus, we can not have an *'Abstract Lane'* defined inside BPMN specifications (and, in case, the definition would collide with the Lane definition). However, we can create a view showing the whole selected Lane and only those elements and flows straitly involved into the sole Lane process. As a secondary consequence, in this way we extend to Lanes the concepts of Private, Abstract and Collaboration types, originally related to Pools. It is possible to see an example of one intraprocess view applied between Lanes in Fig. 5.2(b).

5.2.4 Content-based Classification of Views

In this subsection we define views with respect to their content once they have been computed. In the previous subsections we have depicted three different aspects upon which views can be classified. We have also introduced a taxonomy for view types. Creating a view for a business process means to choose between eight different kinds of views, as summarized in Tab. 5.1. Firstly, we have to choose if we need to select spare elements or if we would like to write a condition which could be applied also on different processes. Secondly, we have to select the wished abstraction level from which we want to extract the view, whether it is the model (graphical) level or the process physical level (the process serialization). At last, consider if the view will concern one process or the view elements will be grabbed from multiple Pools.

Views Types		
Specification	**Level**	**Direction**
Extensional	Model	Interprocess
		Intraprocess
	Physical	Interprocess
		Intraprocess
Intensional	Model	Interprocess
		Intraprocess
	Physical	Interprocess
		Intraprocess

Table 5.1: A summary of the different views types, basing on our classification

For each of these kinds of views we could obtain different results in terms of what the executed view represents. We could have views showing one single element (e.g., a Task or only a Sequence Flow) or the whole diagram, as well as a certain subset of elements, representing spread objects or the choreography between two distinct processes.

Among the others, it is very useful to separate one particular role or entity from the others. BPMN defines a Pool as a representation of one single process with one single participant. Lanes do not have a ParticipantRef attribute nor a ProcessRef, but they represent Pools sub-partitions, and very often sub-partitions

correspond to companies internal roles or entities. So, it is important to have a tool to overhang these characteristics. Figure 5.3 represents the example process we have introduced in Chap. 4 with a Process view highlighted.

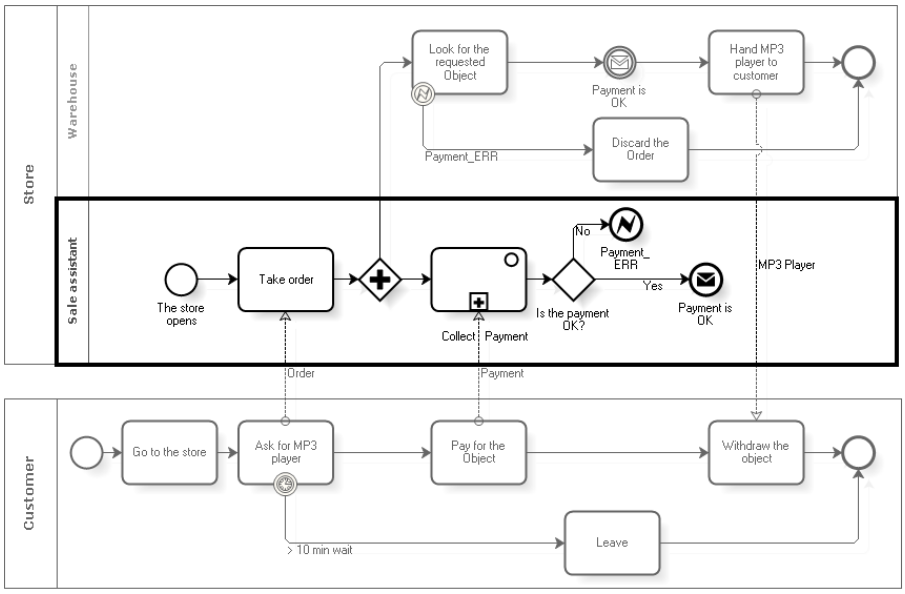


Figure 5.3: A process view example, with the *Sale Assistant* Lane as result

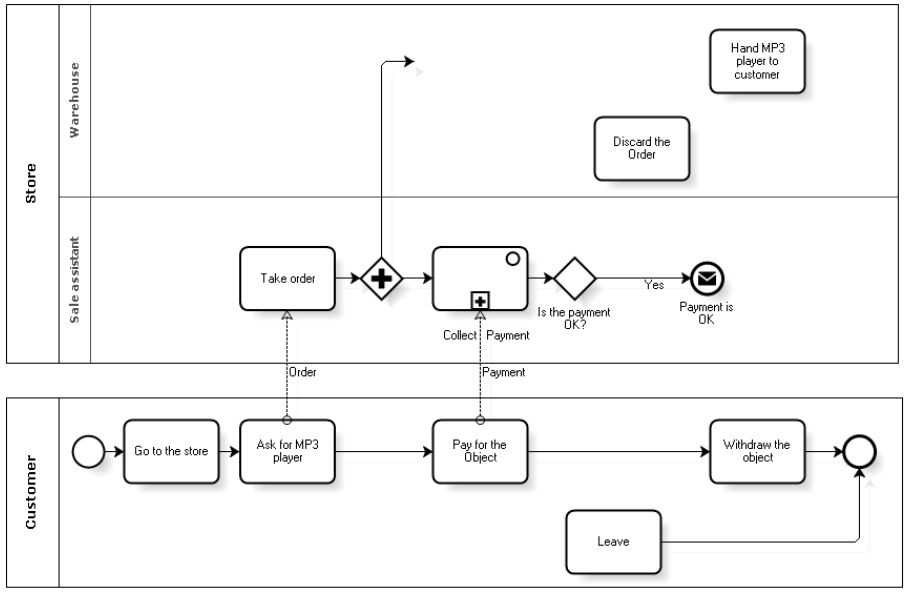
Regardless of the result content, it is important to consider some properties owned by views. In particular, assuming that a view is possibly extracted starting from a BPNF diagram (or, at least, from a correct, complete and compliant diagram), will the view result be likewise in the BPNF (or, at least, correct, complete and compliant)? In general we can easily state that views will not be in BPNF, except for particular and rare cases. Thinking of the aims of views (i.e., to extract information from a process), they probably will not be complete with respect of the starting process. We can also state that if the starting diagram is syntactically correct, also the view will be the same (the graphical representation of elements does not change simply selecting a subset of them from a bigger diagram).

Rather, it is important to evaluate if a view can be considered as a process, and in particular if it is executable – we refer to the BPMN specifications definition of execution semantics. This means to settle if a view is semantically valid with respect of the rules provided by BPMN specifications. We summarize here some main properties:

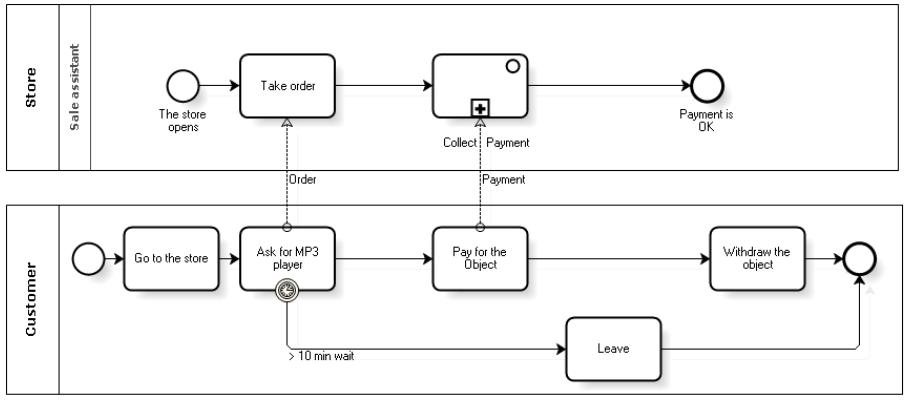
- it could be possible to identify a starting element and one or more possible end points for the process
- it could be possible to follow a flow throughout the process from its beginning right to the end

- endless flows should not exist (either Message Flows and Sequence Flows)
- throwing events should not exist without the corresponding catching events whereas it is not permitted to design branching and splitting gateways without modeling the corresponding merging and joining gateways.

Figure 5.4(a) is an example of a non-executable view, while the Fig. 5.4(b) shows a simple example of a semantically valid view.



(a) One non-executable view



(b) One executable view

Figure 5.4: An example of executable and non-executable views

Working with semantically validated views is a very interesting topic. As we

will discuss in Sect. 5.3, we could have the opportunity to modify views inserting, deleting or updating some elements. Also, we can use views to introduce a users management system inside a business process. These two topics are in the forefront for what concerns BPMN (and, more in general, business processes) analysis and its practical applications. As a matter of facts, both the two BPMN 2.0 proposals provide an extended users definition and administration.

Non-executable views are useful too. They are functional to point out some more specific diagram or process properties. Actually, in most cases views extracted from a business process diagram are non-executable views.

5.3 Updating Views

While in databases one view can be considered updatable if under some conditions, it is possible to translate an *update* statement on a view into unique updates of the underlying base tables, in BPD we have to focus on the diagram structure instead of the data. Adding a gateway or a task in a view could produce bad-formed diagrams or compromise the behavior of the process. So it is a necessary step to define some rules to evaluate whether a view could be updatable or it has to be considered a read-only view. Our intent in this section is to discuss *when* it is possible to execute certain operations on a view, and not *how* to do them. We will provide some examples to better explain the concepts, but they should not be taken as an exhaustive set of update rules.

Considering the relational databases domain, one view can be updated whether the X/Open standard rules are satisfied [32]:

- the FROM clause of the Subquery must contain only a single table, and if that table is a view table it must also be an updatable view table
- neither the GROUP BY nor the HAVING clause is present
- the DISTINCT keyword is not specified
- the WHERE clause does not contain a Subquery that references any table in the FROM clause, directly or indirectly via views
- all result columns of the Subquery are simple column names (that is, there are no arithmetic expressions such as `avg(qty)` or `qty+100`), and no column name appears more than once in distinct result columns.

In particular, no views, based on a `join` or which contain aggregate columns, can ever be updated in most database products. On the contrary, a view which is a Row and Column subset of a single table is considered a safe updatable view. Some databases implementations, like Oracle DBs, introduce also the concept of materialized and non-materialized views. Oracle uses materialized views to replicate data to non-master sites in a replication environment and to cache expensive queries in a data warehouse environment[2].

Summing up, all these restrictions on the definition of views impose that it is possible to precisely map what data has to be updated in the base relation, given the corresponding update in the view.

The BPM domain is different from the DBMS domain especially because BPM focuses on processes description and execution rather than data manipulation. Moreover, business processes are not data tables and thus, obviously, we can not use the views update requirements as they are defined for databases. Dealing with business processes, we do not have operators like joins or aggregate data – at least as they are defined in DBMS domain – nor queries expressed through SQL statements. However, we are interested in defining some conditions by which a view can be considered updatable.

[2] http://download.oracle.com/docs/cd/B10501_01/server.920/a96567/repmview.htm

The update problem for BPDVs can be defined as a disambiguity matter, or, to put it better, whether it is possible to know the final right position of the elements added, deleted or modified in a view inside the underlying BPD. To discuss this issue we borrow some definitions from DBMS domain. Dealing with business processes, it makes sense to use the concept of materialized and non-materialized views. One necessary preliminary remark regards the possibility, each time a non-materialized view is created, to materialize it storing the view content apart from the queries statements.

Like in DBMS domain, non-materialized views are the most common cases of views usage. But sometimes working with one materialized view allows to perform useful tasks, like to add stronger user permissions policies (as we will discuss in Sect. 5.4) or to enable complex processes subsetting as well as to enable users working on independent copies of business processes. Unfortunately, the use of materialized views introduces some issues limiting the update capabilities of such views. As for Oracle DBs materialized views, we have to deal with a two-directions update problem, as it is depicted in Fig. 5.5.

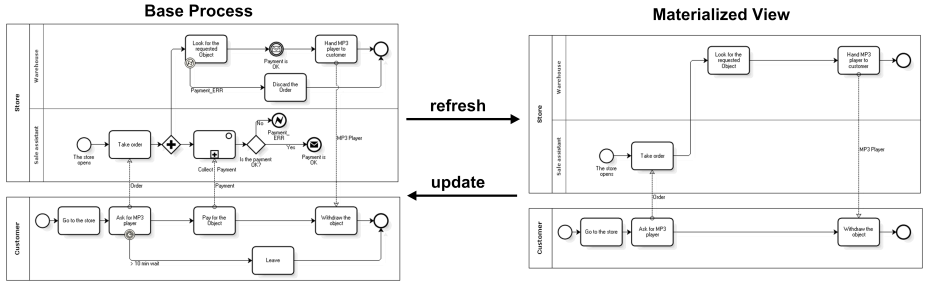


Figure 5.5: The two directions problem with materialized views

Once one materialized view is created, it is stored apart from its generating statements and it is not automatically redrawn whenever the base diagram changes. So, the goodness and the accuracy of one materialized view depend on its refresh frequency. The other direction (from one view towards its base diagram) concerns the 'classical' update scenario: the backwards mapping of views changes in the underlying diagram. While non-materialized views guarantee that we are ever working on one view reflecting the last base diagram version, we can not be sure the base diagram has not changed between the time one materialized view has been created and the view update. So, updating a materialized view can produce an inconsistent base diagram. Non-materialized views, instead, could be considered as abstract snapshots of a subset of base diagram elements with no versioning synchronization problems. However, both materialized and non-materialized views have to satisfy some strict conditions to be updatable: one view update should not produce an unfair base diagram. In the following we will show some examples of views underlying which updates can be backwards mapped and which ones determine inconsistency or ambiguities when backported to the base diagrams.

5.3.1 Backwards Updatable Views

In this subsection we investigate more in detail the requirements needed by one updatable view to introduce changes which can be transferred backwards on the base diagram. Every view can be freely manipulated but not all the modifications can be reported back to the original business process. The two main problems with views updates can be classified as:

- Syntactic correctness (or disambiguity matter)
- Semantic soundness (and BPMN specifications compliance)

As we will detail in the following, adding or removing one or more elements entail to be able to exactly determine the right final positions of such elements inside the base diagram. We suppose that we can precisely position all the new elements or remove the deleted ones. We have also to ensure that the updated base diagram is still semantically correct and compliant with the adopted specifications – in this case, BPMN specifications. Update operations can be differentiated as *insert*, *delete* and *update* tasks.

Syntactic correctness. If we add one new element to one view, we must be able to find inside the base diagram the right final position of such element. For instance, looking at the Fig. 5.6, we consider the view showing only Tasks **A** and **C** from the base diagram and not the Task **B**.

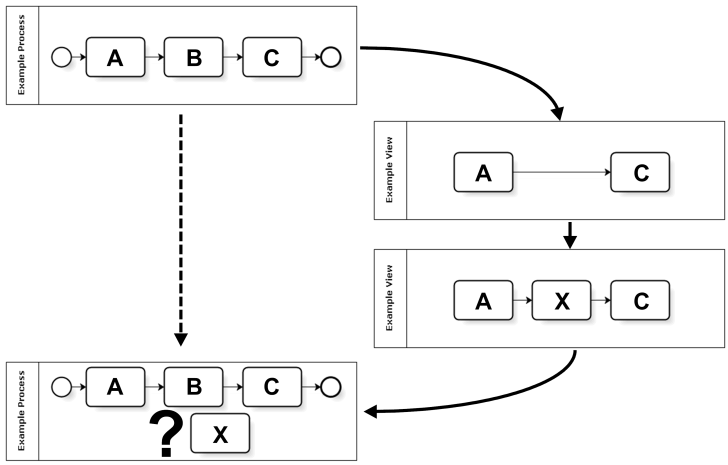


Figure 5.6: The insert operation could cause a disambiguity matter

If we add the Task **X** inside the view, where does such Task have to be positioned inside the base diagram? If we do not have enough information, we can not determine if it has to be inserted between **A** and **B** or between **B** and **C** or even if it could be executed in parallel with the Task **B**. Moreover, it could happen that Task **X** was already defined but our permissions denied us to visualize it. To add Gateways or Events is even more complicated, because both of them could

noticeably affect the number and/or the paths of Sequence and Message Flows which are defined inside the base diagram.

Quite similar to the problem of inserting new elements, also to remove one or more objects from a view is possible only if we know how to connect all the remaining objects. In Fig. 5.7 we try to remove the Task **C** from the view. In the base diagram we had one Message Flow ending to Task **C**. Now we have to attach the Message Flow to another Task or to remove also the Message Flow. But there is a lack of information to help us to decide how to modify the base diagram.

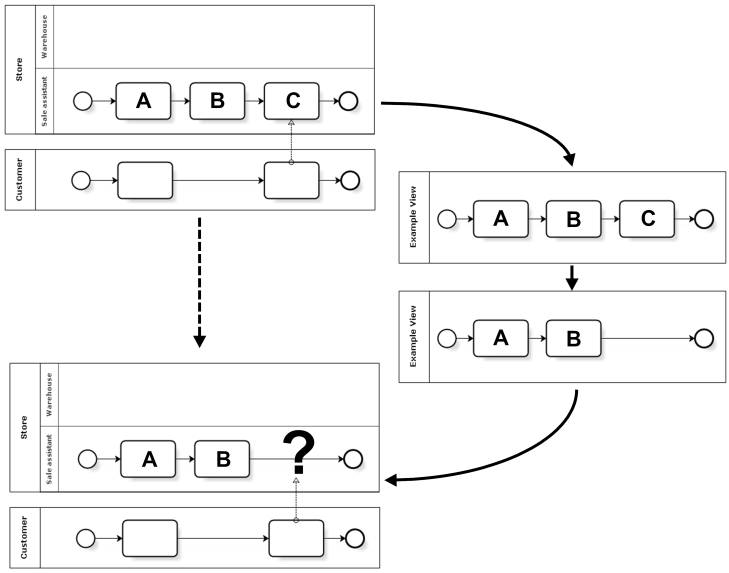


Figure 5.7: The delete operation on a view could produce non-ending paths

We show another example with more complicated conditions to be provided in order to backwards map the changes made inside the view. Consider the view in Fig. 5.8: we remove from the view the Task **C** as in the previous example, but in this case the Task **C** is part of one Parallel path starting from the Gateway.

This view shows only two of the three possible paths defined inside the underlying diagram. Removing the Task **C** from the view makes the Gateway unnecessary, and thus we decide to remove also the Gateway. But, when we try to backport the changes we have to deal with the Task **B** which was hidden in the view and had to be executed in parallel with both Tasks **A** and **C**.

The update of views objects is generally not affected by the disambiguity matter. It is possible to change the label of one flow or the text description of one Task without this affecting the behavior of the underline diagram. Elements are all referenced by their unique ID value, and, thus, to change a textual description or some descriptive attributes values does not involve paths redesign or reallocation of elements references. Moreover, disambiguity matter involves the necessity to find the right position of the object after a view adds or removes elements.

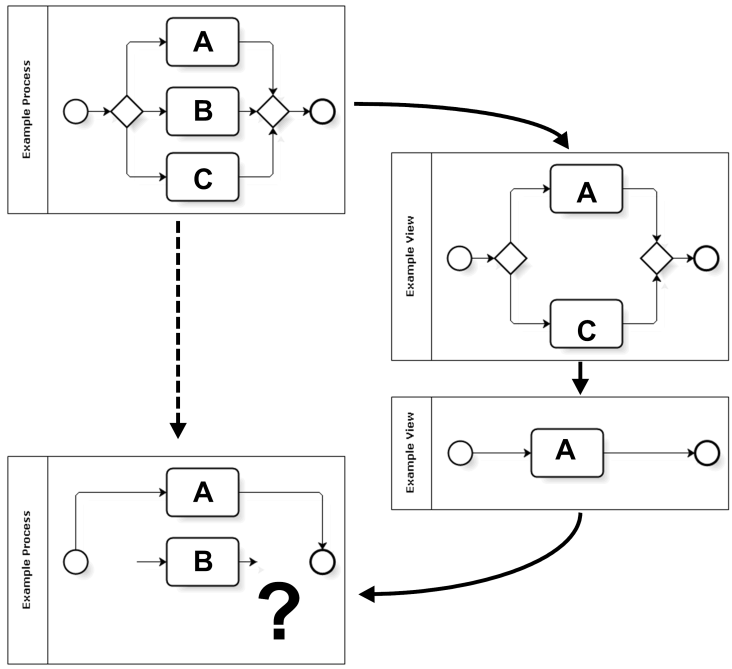


Figure 5.8: To remove Gateways could create underdetermined diagrams

To update elements without adding or removing object does not change the base diagram. There could be updates which substitute one element with another one of the same type changing only its specialization. For instance, we can update one view changing one Task with one Sub-Process, one Intermediate Message Event with one Intermediate Timer Event, or even one Parallel Gateway with one Data-Based XOR Gateway. These changes do not create disambiguity matters because we substitute one element with another element, even if they could produce semantically wrong diagrams.

Definition 1 (Syntactical Correctness)**.** *One update operation on a view is possible if and only if it produces a syntactical valid base diagram.*

Semantic soundness. Even if we can correctly find the right placement for every inserted element, or redraw connections between elements once an object has been removed, we have to consider if the updated base diagram is still semantically valid.

The semantic correctness of one base diagram is not completely independent from the syntactic correctness. In some cases, like the insert of one new Task, the semantic of a single elements could help us determine if the Task should precede or succeed another Task. The Second Round of the design methodology we presented in Chap. 4 could help modelers to find the right position of new elements. However, it is not guaranteed that each element could be rightly positioned, and

thus the update is not valid: the updated base diagram should not contain ambiguities. If one Gateway is inserted as a new element, we have to consider all the related consequences. For instance, if one flow becomes part of one Exclusive Gateway, there could be the possibility not to execute some Tasks. Adding a Sequence or a Message Flow could determine deadlocks or cycles. Again, we can easily insert one Intermediate Event between other elements. But, if the Intermediate Event is of type Cancel, than the base diagram should have somewhere one End Event of the same type Cancel. If only the Intermediate Event is inserted in one view and no End Event is declared, the resulting base diagram will not be semantically valid.

Removing one or more objects from the view could cause more problems than the insert update. Consider Fig. 5.9: we can remove the Start Event from the view because there is not an End Events. But the base diagram had two End Events, one of them hidden inside of view. BPMN specifications state that if one or more End Events are declared, one Start Event must be declared. So we get a base diagram semantically wrong.

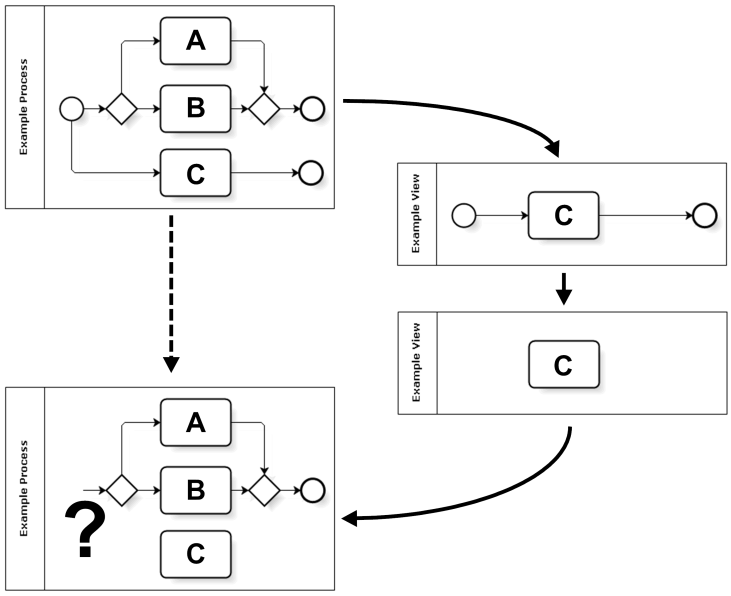


Figure 5.9: The base process is no more correct after the update is backported

Moreover, deleting the Start Event causes the other two paths between the Gateways not to have a clearly stated beginning. If no Start Event is declared, the instantiation semantics of BPMN provide that the process execution must start from one Task having no incoming Sequence Flows. Tasks **A** and **B** are preceded by one Gateway and, thus, they cannot start the execution. This is an indefinite, ambiguous scenario, and it is not a semantically valid base diagram.

From the semantic soundness point of view, updating one element changing some of its characteristics could determine an invalid base diagram. Looking at

the example view of Fig. 5.7, we assume that to design the base diagram we follow the design methodology of Chap. 4 providing a rule for which the Task **A** must precede the Task **B**. If we invert the labels of the Tasks **B** and **C** inside the view, we get an erroneous base diagram. Besides, in the base diagram of this example we have one Message Flow ending to the Task **C**. Inside the view we do not see the Message Flow, but when we map backwards the update we discover that the Message Flow points to the wrong Task (the Task **B** rather than the Task **C**).

Definition 2 (Semantic Soundness)**.** *One update operation on a view is possible if and only if it produces a semantic valid base diagram.*

Summing up, we can state that one view can be considered updatable if both Definitions 1 and 2 are satisfied. That is, we can update one view if we can precisely map backwards in the base diagram all the updates and the resulting base diagram remains syntactically and semantically correct.

5.4 Views for BP Access Control Policies

Having introduced the BPDV definition in the previous sections and the conditions under which one view can be updated, in this section we aim to explore the possibility to use BPDVs to express users' access control policies.

BPMN considers users only as one of the processes description aspects. BPMN represents users as processes Participants, that could be specified as Roles or Entities. Business processes user management is a very relevant asset. As a matter of facts, the two BPMN 2.0 proposals aim to extend the users definition adding some improvements. However, this users definition is related only to the processes description, and it is provided in terms of associating one or more Participants to their processes.

Nowadays the attention has been only focused on the business processes static description (modeling). The main focus is even more moving towards the processes execution. So, it is important to pay close attention not only to the users as Participants of a process to be described, but also as final users (either roles or entities) who access the process description to perform some actions (typically, to read, to execute or to update diagram pieces).

At this moment it is not possible to use BPMN to describe users accessing a business process diagram. BPMN was originally published to describe and to model business processes. BPMN 2.0 will hopefully provide a better support for Participants definition. One mechanism to express how a final user – a reader – could access one diagram is not explicitly foreseen.

We aim to fill this gap providing a tool which can be used to establish

- the **user** who is accessing the diagram
- which **view** the **user** is accessing to
- what **action** the **user** is performing with the **view** he is accessing
- whether the **user** can access to that **view** to perform that **action**

With BPDVs we can directly represent the first two statements, creating views from a base diagram and associating those views with different users. To establish the actions set which one user can perform on a determined view we need to extend BPDVs with other tools and formats, like the Platform for Privacy Preferences (P3P) for Web-services scenarios – as we will discuss in the following Chap. 6 – or RBAC for a wider range of domains. Thus, it becomes possible to use also in the business process management domain concepts like *owner*, *group*, *actions*, *granting*, and so on. For instance, the owner of one view is the user who creates the view. The owner can do everything with his views, including to spread to other users his permissions on a view. The owner could create a sub-view starting from one view (if the view structure makes the user able to do so) giving to other users update privileges strictly on that sub-view.

The permissions granting can concern the same view level if we want to emphasize the different processes Participants (for example, using the example introduced in the Chap. 4, the Store can grant to the Warehouseman some permissions on the Tasks he usually performs, denying also to view or modify all the other

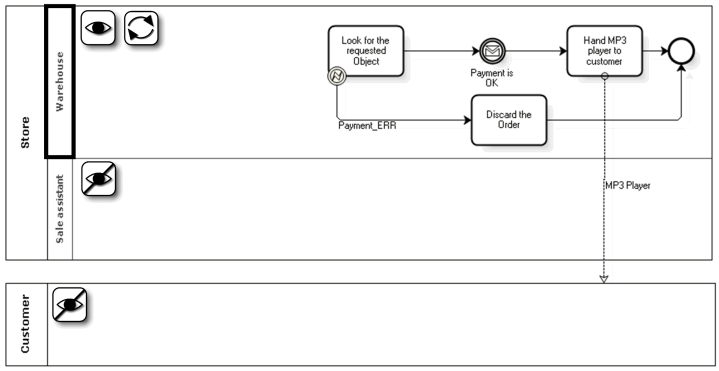


Figure 5.10: The Warehouseman's view with access permissions granted by Store

processes, as in Fig. 5.10), or it can have a hierarchical structure (as it is possible to see in Fig. 5.14), if the user is interested in changing the abstraction level.

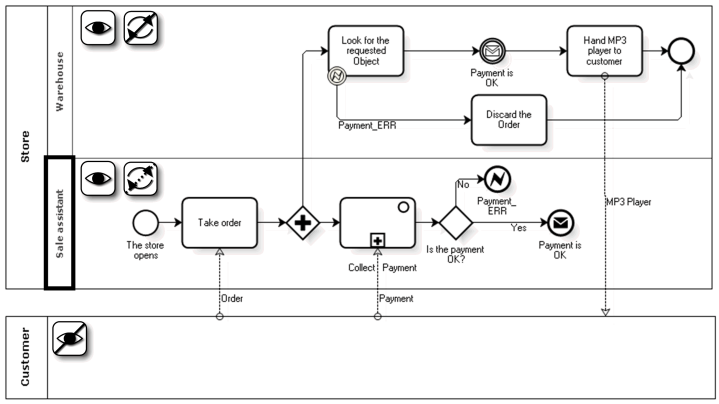


Figure 5.11: The Sale Assistant's view with access permissions granted by Store

Starting from the Customer/Store example, consider the Sale Assistant's view depicted in Fig. 5.11. It can be used to exemplify the permissions related to the Sale Assistant view. In this example the Sale Assistant can completely see the Lane he belongs to and he can modify some Tasks. He can see the Warehouseman Lane but he can not update that Lane. He can not visualize the Customer process at all. We can apply the same method to both the Warehouseman's and the Customer's views. The Warehouseman can visualize and update his Lane but he can not see or modify the Sale Assistant or the Customer's Lanes and processes. The Customer can see the Sale Assistant's Lane and some Tasks of the Warehouseman's Lane, but he can not modify them. These three views are shown in Fig. 5.13 using a non-standard graphical notation as a BPMN extension to display the different permissions users have on views. An explanation of such

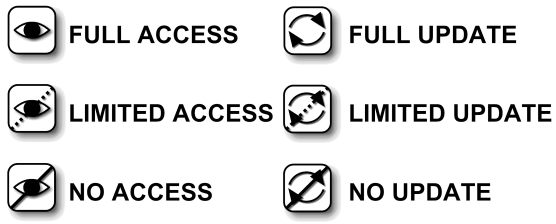
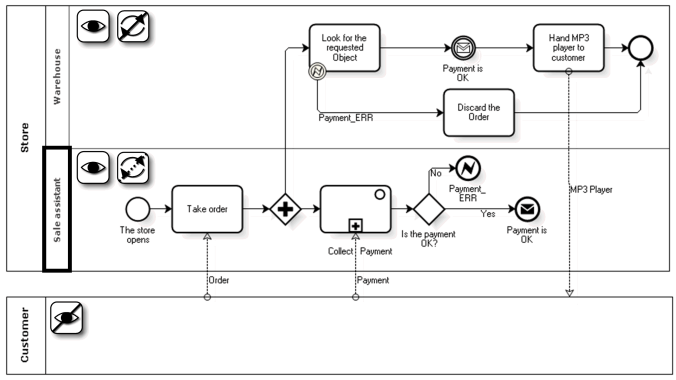


Figure 5.12: A legend for the Access Permissions symbols

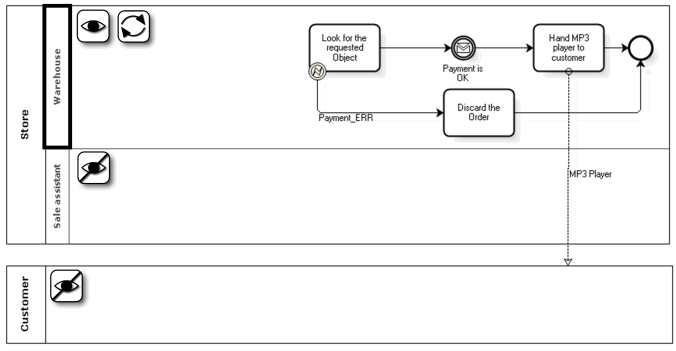
symbols is provided in Fig. 5.12.

Using the same example, the scenario depicted in Fig. 5.14 can be used to describe the general process Store which gives to the Warehouseman the permissions to manage his Lane (that is to manage his view). Tasks can not be deleted or modified, but the Warehouseman can better detail the single actions to be taken to accomplish each Task assignment. The warehouseman could also delegate to other users part of the whole Task. If we continue to cascade with descriptions and detailed specifications we can obtain a hierarchical views structure.

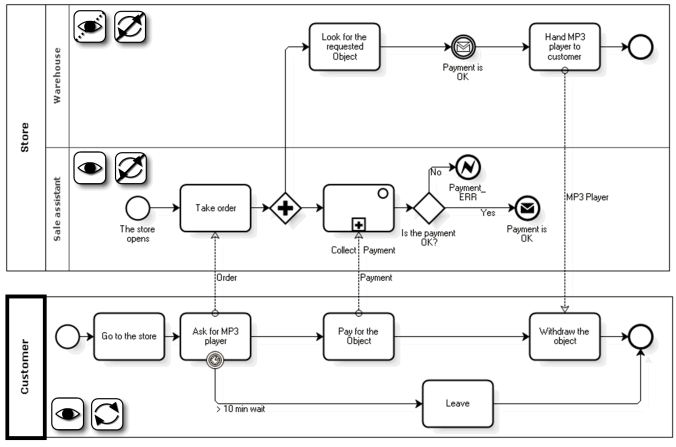
Working with even more complex business processes the access control policies management becomes a very useful and important asset. This aspect surely complicates processes executions capabilities, because it becomes necessary that all the users have execution permissions over the view. However, it is possible to share one process model (one BPD) freely with all the internal employees but also with the public, granting access permissions and avoiding the modeler to redesign one different process for each reader, which introduces redundancy and synchronization risks.



(a) Sale Assistant view and permissions



(b) Wharehouseman view and permissions



(c) Customer view and permissions

Figure 5.13: Users access and update permissions

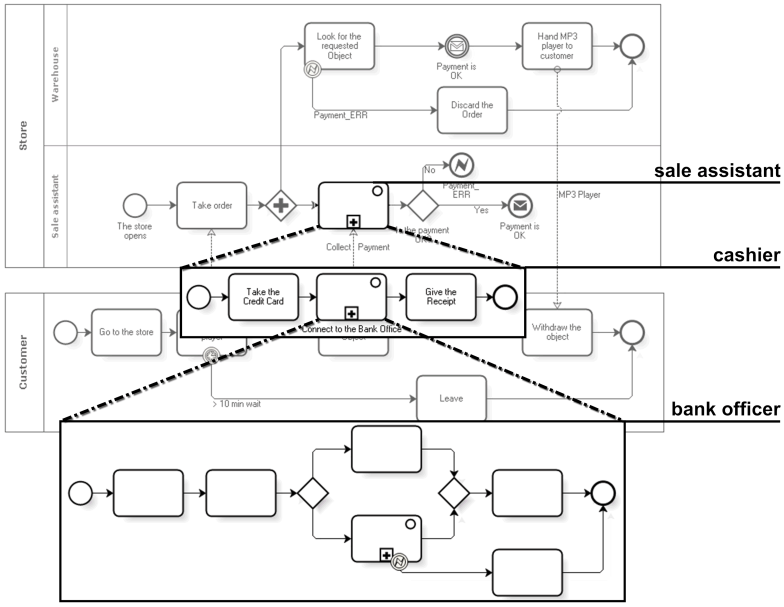


Figure 5.14: A hierarchical views model involving different users for each level

Chapter 6

Integrating Privacy Policies into Business Processes

The adoption of BPMN as a standard allows companies to define complex business processes possibly encompassing different administrative boundaries and requesting non-public data whose access is regulated by security-driven policies. In particular, privacy-related user data represent a relevant asset for companies and administrations. As such, in the last years, several efforts have been made in order to provide mechanisms to express (and in some cases, enforce) privacy policies which protect such data. One of the most well-known efforts in this direction is the P3P privacy policy description language [7]. P3P permits to represent privacy policies in an XML tree, on the base of an XML-Schema model, which can be published by service providers to give users the capability to check automatically if the policies accomplish with the user preferences. Therefore, it becomes a relevant, non-trivial issue to check whether a given complex business process (describing how processes interact and what data items they access) is compliant with a stated privacy policy (describing what processes are entitled to access which data items, provided that the corresponding purposes and obligations have been stated).

Other works about integrating privacy policies with business processes have been published since 2002 [20, 1, 4] even though the first work proposing an algorithm to verify P3P policies on BPEL tree was published in 2006 [24]. Moreover, in [31] an approach to extract RBAC models from BPEL processes for the role engineering process is presented.

In this chapter we present a succesfully application of the BPeX model addressing the above mentioned issue by presenting a single framework in which both a business process and a corresponding privacy policy can be expressed and the compliance of the former with respect to the latter can be checked. We accomplish this issue by expressing both business processes and privacy policies in suitable XML formats and then proceed to check their compliance. We assume that privacy policies are expressed in P3P format, while we use BPeX for business processes serializations.

Having presented in a single, coherent framework both business processes and

privacy policies, we then go on defining the procedures for checking the compliance of a BPeX-based business process with respect to a P3P-based privacy policy. Such procedures, relying on the unified XML-based format employed for business processes and privacy policies, are implemented using standard XML query languages, such as XQuery/XPath.

6.1 P3P: Platform for Privacy Preferences

A privacy policy can be intended as a non-trivial extension of a classical security policy (stating which users can access what data items) in which are explicitly stated what are the purposes of users wishing to access data and what are the intended obligations the user has to perform, in order to have the privacy policy fulfilled. Over the web, these features are (partially) provided by P3P, a widely-supported privacy policy definition language. The *Platform for Privacy Preferences* (P3Pv1.1), at present a World Wide Web Consortium (W3C) Working Group Note, is the most significant effort to enable users to be informed about Web site privacy practices on their private data. P3P policies apply also to Web-services applications. P3P user agents alleviate users from reading privacy policies, automating decision-making based on the Web site practices when appropriate. A user can be informed about privacy policies before he releases his personal information. P3P provides a mechanism to express privacy policies in a standard machine-readable XML format known as a *P3P policy* [7]. Unfortunately, as it is possible to see in Fig. 6.1, the P3P working group at W3C closed because *"there was insufficient support from current Browser implementers for the implementation of P3P 1.1"*.

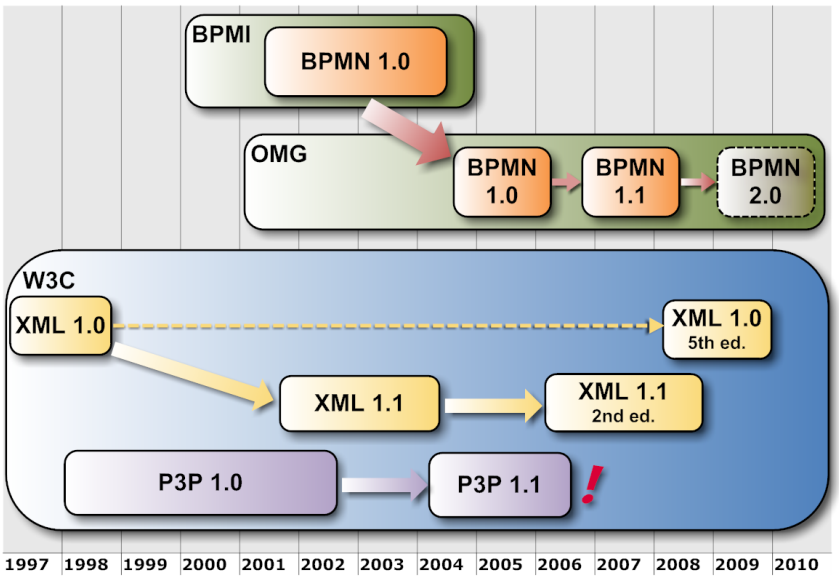


Figure 6.1: A time line with BPMN and P3P development processes emphasized

P3P policies consist on a sequence of `STATEMENT` elements, and each `STATEMENT` includes six sub-elements:

- `PURPOSE` represents the aims for data processing. There are 12 different purposes (e.g. *current*, *admin*, *contact*, *telemarketing*).
- `RECIPIENT` is the legal entity, or domain, where data may be distributed. There are six different recipients (e.g. *ours*, *same*, *public*).

- `RETENTION` is the type of retention policy in effect. There are 5 types of retention: *no-retention*, *stated-purpose*, *legal-requirement*, *business-practices*, *indefinitely*.
- `DATA-GROUP` describes the data to be transferred or inferred. It contains one or more `DATATYPE` extension elements. Datatype extension elements are used to describe the type of data that a recipient collects.
- `CONSEQUENCE` and `NON-IDENTIFIABLE` are optional sub-elements.

A P3P policy provides *always*, *opt-in* and *opt-out* values for the `REQUIRED` attribute that `PURPOSE` and `RECIPIENT` may have. P3P has also predefined types of data items and policies can be expressed using `CATEGORIES` of data items too. These elements are located inside `DATA-GROUP` nodes. Examples of `CATEGORIES` include *physical*, *online*, *purchase*, *computer*, *demographic*, *health*, *location* elements.

The P3P language does not have the expressive power to state full-fledged privacy policies. In particular, P3P does not explicitly represent general purpose obligations, but only the particular case of retention is taken into account. However, the core subset is rich enough to achieve the minimal requirements a privacy policy should have. P3P can express only `<Recipient>`, `<Data>` and `<Purposes>` elements of a privacy policy definition, with a direct representation. To the contrary, P3P has not enough meaningful elements to represent `<Actions>`, `<Conditions>` and `<Obligations>`. It is possible to use some of the values of P3P notation to deduce missing constructions or to extend the original standard.

6.2 P3P Policy Enforcement

Currently, the main use of the P3P language is for web services providers, which can host policies in their servers let users opt whether using the service provided or not. Some browsers can access P3P policy document and warn users if a server policy does not accomplish the users' preferences.

BPMN (and BPeX, consequently) can easily be adopted to describe business processes whose tasks have to follow a given privacy policy. Extending the BPeX model in order to add P3P support permits users to test if a web-enabled business process is compliant with a given privacy policy. For example, a web service provider which asks user for a credit card number to perform a given task could be in contrast with the privacy policy which does not allow to ask for personal information.

In our approach, we will extend as less as possible BPMN notation in order to keep the main requirements unchanged. Notice that P3P does not implement a full privacy policies tuple unlike, for example, the RBAC model. This is because P3P is a web-oriented standard. The main aspects we will use to extend BPMN notation are Entity, Purposes, Access, Data-group and Recipient.

For some of these elements (Entity, Data-group) we will use BPMN native attributes, extending the notation to better explain and represent the values of P3P elements. For the Access element we will add a new attribute to a BPMN Process element. For the other elements we need to redefine or tune BPMN elements modifying some attributes or adding new ones. These modifications will be mapped also in BPeX to achieve a full XML linearization of BPDs with P3P statements. We have developed also some simple procedures to perform validation tests between BPeX and P3P policies trees. Some examples will be shown in the following using W3C XPath queries.

6.2.1 Motivating Example

The example we introduce in this section sketches the environment we are interested in. We start considering a web-oriented business process that we represent with BPMN. Then we translate the BPD into a BPeX representation. In the following subsection we will illustrate how to integrate P3P policies and BPeX documents. Finally, we present some excerpts of BPeX code enriched with privacy policies and the algorithms to enforce the process policy.

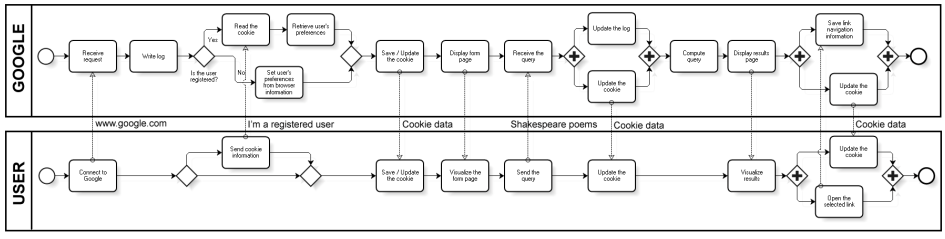


Figure 6.2: BPMN representation of a user connecting to a search engine to perform a query

For our running example we use a classical scenario of a user connecting to a search engine to perform a query. For the sake of clarity, we opt for Google and its privacy policies freely available on-line at Google Privacy Center[1]. Figure 6.2 shows the BPMN model of the Business Process we choose to investigate and Listing C.1 (available in Appendix C) is a simplified part of its BPeX linearization. The text of the search engine privacy policy has been taken from the on-line version and the Listing C.2 (Appendix C) is related to its P3P form. This example takes care of most of the Google Privacy Policy conditions. For example this process shows the cookies exchange between Google and the user's browser or the data which are stored in a log file. The process models the alternatives which can be followed depending on the registration status of the user to Google services. In some cases along this chapter we will show some excerpts taken from this process. A bigger version of the Fig. 6.2 can be found in the Appendix C.

6.2.2 P3P Representation Inside BPeX Code

Now we summarize the formalisms we use to represent P3P clauses inside the BPeX code, investigating as much in detail as possible each correspondence.

Entity. This element refers the legal entity making the representation of the privacy practices. There are only two elements in BPMN which can be used to map the Entity: the BPD and the Pool. We can not use the Process element because there may be Pools without a related Process (Black Boxes). Nonetheless, also in these cases a Pool represents a subject involved in the BP. Between BPD and Pool elements we choose to use the latter, because a BPD is a set of all the processes pertaining the BP, while a Pool represents a single participant (i.e., a single `Entity`). P3P binds some values to be present in a policy. A P3P `Entity` must hold the `orgname` attribute and one of the following categories of information: `postal`, `telephone`, `email`, `URI`. For the sake of simplicity we extend the `Name` attribute of Pools (i.e., `Pool/Name`) adding a new sub-tree starting with the `<P3PExension>` node, father of an `<Entity>` node. The Entity node imitates the P3P `Entity` nodes structure, with the same constraints. The new node `P3PExtension/Entity/orgname` substitutes the old Pool Name. Adding the same P3P `Entity` subtree to the `Pool/Name` attribute makes easier to compare values and enforces this policies element: there should be a direct correspondence between the two nodes structures, as depicted in Figure 6.3. This is a true advantage in using BPeX model respect to the original BPMN proposal, because with the latter it is not possible to make a comparison node-to-node.

Access. The P3P Access element represents the ability of the individual to view identified data and address questions or concerns to the service provider [7]. In this case, BPMN does not have an element near to the Access, but each Pool holding activities and flows has also a relationship with one Process. We add to the Process element a new attribute `<P3PExtension>` having as child the `<ACCESS>` element. Possible values of `<ACCESS>` element are those provided by P3P standard.

[1] http://www.google.com/privacypolicy.html

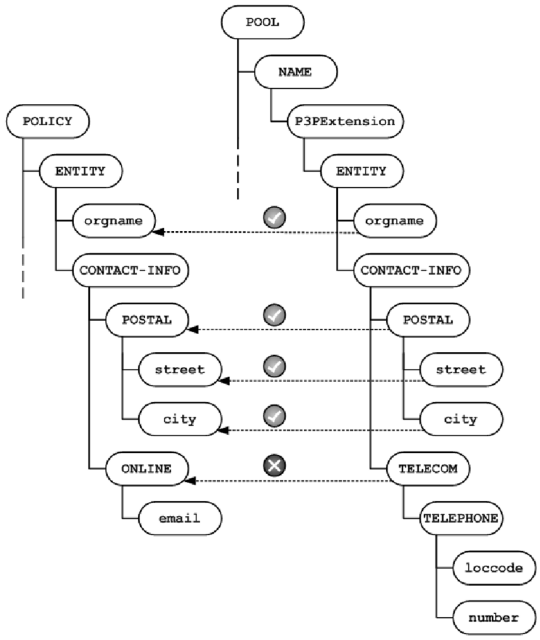


Figure 6.3: A comparison between P3P and BPeX Entity elements

Purposes. Every 'Common Graphical Object' (i.e., all the graphical objects which may appear in a BPD) has a `Categories` attribute, defined as follows: "The modeler MAY add one or more defined categories which can be used for purposes such as reporting and analysis". Thus, the use of this element as a container for the P3P Purposes element does not require any further adjustment unless a boolean attribute, named `IsP3PPurpose`, to better define the purposes domain. All the BPMN elements except for BPD and Process (which do not have a graphical representation) have the Categories attribute, so we can define, for every element, which purpose it is designed for.

Data-group. This is the most critical issue because P3P is very rich of details about data while BPMN provides only an Artifact named `DataObject` to represent all the possible kinds of data. To describe as well as possible the information exchanged through the activities and the flows of a BP we use the Name attribute of the DataObject element to specify what kind of data an entity or an user is working on. Like so we have done to map Entity element into the `Pool/Name` attribute, we extend the `DataObject/Name` attribute with a `<P3PExtension>` node, containing the trees provided by P3P Data-Group element. In addition, BPMN DataObjects have a `RequiredForStart` boolean attribute, that can be used to map the P3P `always`, `opt-in` and `opt-out` values for purposes and recipients.

Recipient. This element contains one or more recipients of collected data. It is the legal entity, or domain, where data may be distributed. The mapping of the `Recipient` element is a bit more complex. The best place to attach the Recipient data is a `Message Flow` (which represents the messages exchanged between different Pools – and, thus, different Entities). Unfortunately MessageFlows do not have a direct attribute where to specify the Recipient constraints. It is unnecessary to control messages exchanged inside the same Pool, between different Lanes: we suppose that an Entity can freely share data with its offices or internal employees. Again, we are not interested in investigating where data come from (typically, in BPMN diagrams, through Message Flows) – it is the sender Entity that has to adhere to its privacy policy. We need to ensure that data collected from an enterprise are the same of those declared in its privacy policy. What we can not express in BPMN is the affiliation domain of the messages targets. P3P does not need to know the target entity data, but only if the target, for example, has the same privacy policies or if it is the legal entity following the practices, and so forth. To add this kind of information, we extend the `Target` node of a `Message Flow` with an attribute `P3PRecipient` expressing the P3P values provided for the `Recipent` element.

6.3 The Compliance Checking Procedures

Our goal is to check whether a BPMN diagram, representing a web-enabled business process, is compliant with a P3P privacy policy. Thus, as discussed in the previous section, we have enriched the BPeX XML-based BPMN representation with some P3P-like attributes. Now, we provide checking procedures in order to verify such compliance. The tests we are interested in focus on the presence of the same attributes either in BPeX and in P3P trees. P3P notation is not used to express the values collected for each instance of the service provided. Thus, the tests will not cover the correspondence between the values which can be performed only when a process has been executed through log analysis or using a monitor.

We start assuming that each *Pool* represents an *Entity*, and thus we make the tests on *Entity* and *Access* between the *Pool* attributes and respectively `POLICY/ENTITY` and `POLICY/ACCESS` attributes. All the other tests are performed for each P3P `STATEMENT` clause, and focus on: what kind of data the process works on, how the process uses collected data and with who an entity shares collected data with. In general it is not true that every `STATEMENT` element corresponds to one single Pool: a Pool references one Policy but it may have more than one Statement.

The diagram shown in Fig. 6.4 relates to our Google example: there is one Policy with altogether four Data-Ref elements, three Purposes and two different Recipients. This example shows how different Statements can act using different triples `<Data-Groups, Purposes, Recipients>`. P3P standard specifies that each Statement must hold one Data-Group node and may have more than one Purpose or Recipient expressed. In our example, the *Statement A* uses all the four `<DATA-REF>` values as Data-Group for the Purposes `<admin><develop>` sharing data with Recipient `<ours>`; the *Statement B* instead uses only two of the `<DATA-REF>` elements as Data-Group for the Purpose `<pseudo-analysis>` disclosing data to `<unrelated>` Recipients.

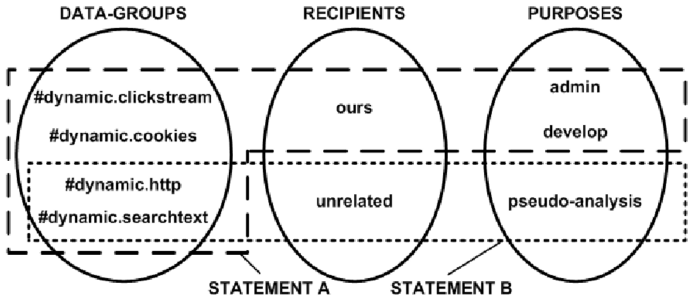


Figure 6.4: Data-Groups, Purposes, Recipients relationships

For the Statements verification we ensure firstly the correctness of the three fields separately and then the accordance between them and against the referential P3P policy.

6.3.1 Policies Enforcement

We introduce now a high level description of the checking procedures. For the sake of simplicity, the procedures presented here do not consider the strings manipulation needed to extract the P3P clauses from the attributes value. This aspect will be shown later on, when we will illustrate an XPath implementation example of these procedures. Considering that each one of the algorithms verifies a different aspect of the policy, then all of these have to be executed to enforce the privacy policy in its entirety.

ENTITY verification. The Listing 6.1 shows the algorithm to enforce the policy of a business process focusing on the Entity verification. This control applies on every Pool (row 1). The first condition (row 2) verifies if the `P3PExtension` node, child of `Pool/Name`, exists: if not, an error occurs (moreover, this implies that the diagram is not compliant with BPMN specifications, because the new node `Name/P3PExtension/orgname` corresponds to the original `Name` value). The core of the algorithm compares the `P3PExtension/ENTITY` subtree with the `P3P:POLICY/ENTITY` one (row 5) like in Fig. 6.3.

```
foreach (Pool/Name PN ∈ BPD) do {                                    1
  if (PN/P3PExtension/ENTITY == ∅)                                   2
   then ‘‘Error’’                                                    3
  elseif (PN/P3PExtension/ENTITY ≠ P3P:POLICY/ENTITY)                4
   then ‘‘Error’’;                                                   5
  else ‘‘OK’’; }                                                     6
```

Listing 6.1: The ENTITY enforcement algorithm

ACCESS verification. The Access verification algorithm (see List. 6.2) is quite similar to the Entity one. It differs from the latter especially for the first condition (row 1) in which there is a check to assure that the Pool is not a Black Box: otherwise, it can not have an Access attribute because the content of the Pool is hidden and users can not access their data.

```
foreach (Pool/Process PP ∈ BPD | PP ≠ ∅) do {                        1
  if (PP/P3PExtension/ACCESS == ∅) then ‘‘Error’’;                   2
  elseif (PP/P3PExtension/ACCESS ≠ P3P:POLICY/ACCESS)                3
   then ‘‘Error’’                                                    4
  else ‘‘OK’’; }                                                     5
```

Listing 6.2: The ACCESS algorithm

PURPOSES verification. In this case (see List. 6.3), firstly we consider only the BPMN Flow Objects, which will be indicated with the abbreviation `FO`. We argue that Swimlanes, Artifacts and Connecting Objects should not have a related Purpose. Secondly, we check if the Categories element has the required boolean attribute. Finally, we compare Categories children with all the Purpose children. Notice that between POLICY and PURPOSES at row 6 there are two slashes '`//`' to show, using the XPath syntax, that Purposes nodes are not direct Policy children but they are Policy descendants through the Statement node.

```
FO := FlowObjects;                                                    1
foreach (Pool P ∈ BPD) do {                                           2
 foreach (fo ∈ FO) do {                                               3
  if (fo/Categories@IsP3PPurpose == ∅)                                4
   then ‘‘Error’’                                                     5
  elseif (fo/Categories ⊈ P3P:POLICY//PURPOSES)                       6
   then ‘‘Error’’                                                     7
  else ‘‘OK’’; } }                                                    8
```

Listing 6.3: The PURPOSES enforcement algorithm

DATA-GROUP verification. Similarly to Entity and Access verification, to enforce Data-Group elements requires to check if the P3PExtension node has been declared and successively if its values fall into the set of every Statement's Data-Group (Listing 6.4).

```
foreach (DATAOBJECT DO ∈ BPD) do {                                    1
 if (DO/NAME/P3PExtension == ∅) then ‘‘Error’’                        2
  elseif (DO/NAME/P3PExtension ⊈ ↩                                    3
                    P3P:POLICY/STATEMENT/DATA-GROUP)                  4
   then ‘‘Error’’                                                     5
  else ‘‘OK’’; }                                                      6
```

Listing 6.4: The DATA-GROUP enforcement algorithm

RECIPIENT verification. To determine if MessageFlows are compliant with their related policies, it is necessary to control if the value of the P3PRecipient attribute is one of those declared as policy Recipient.

```
foreach (MESSAGEFLOW MF ∈ BPD) do {                                   1
 if (MF/Target@P3PRecipient == ∅) then ‘‘Error’’                      2
 elseif (MF/Target@P3PRecipient ⊈ ↩                                   3
      P3P:POLICY/STATEMENT/RECIPIENT) then ‘‘Error’’                  4
 else ‘‘OK’’; }                                                       5
```

Listing 6.5: The RECIPIENT enforcement algorithm

Listings 6.3, 6.4 and 6.5 need to be executed at Statement level, while Listings 6.1 and 6.2 at Policy level. To enforce the whole process against privacy policies, we need to evaluate once the latter and for each Statement the former. If all the tests are passed we can claim that the process is compliant with the privacy policy.

Listings 6.6 to 6.10 are the XPath 2.0 / XQuery implementations of the conceptual algorithms provided to check the compliance of the P3P clauses. The BPeX code (which represents the process) and the policy code are marked with different namespaces. In the examples we omit the `bpex` namespace for a reason of space. To compare two node-sets we employ the XPath 2.0 function `fn:deep-equal`, which assesses whether two sequences are deep-equal to each other. To be deep-equal they must contain items which are pairwise deep-equal; and to be deep-equal two items must either be atomic values that compare equal, or nodes of the same kind, with the same name, whose children are 'deep-equal'. We use also the `some...satisfies` construct to better express the subset relationship, as in List. 6.8. For the sake of simplicity this code uses the P3P 1.0 node structure `POLICIES/POLICY/ENTITY/DATA-GROUP`. Take care that the version provided in

the XPath Listings has been simplified. Some fine tuning and a lot of tests would be necessary in order to properly execute the queries.

```
if (//Pools/Name/P3PExtension/ENTITY)
 then fn:deep-equal(//Pools/Name/P3PExtension/ENTITY,  ↩
                p3p:POLICIES/p3p:POLICY/p3p:ENTITY)
```

Listing 6.6: The XPath version of the Entity algorithm

```
let $PP := //Process[@ID = //Pool@ProcessRef]
if ($PP/P3PExtension/ACCESS)
 then fn:deep-equal($PP/P3PExtension/ACCESS,  ↩
                p3p:POLICIES/p3p:POLICY/p3p:ACCESS)
```

Listing 6.7: The XPath version of the Access algorithm

```
if (//DataObject/Name/P3PExtension)
 then some $dg in p3p:POLICIES/p3p:POLICY/p3p:STATEMENT/p3p:DATAGROUP
  satisfies $dg = //DataObject/Name/P3PExtension
```

Listing 6.8: The XPath version of the DataGroup algorithm

```
for $p in {//Pool}
 let $fo := $p//(StartEvent|IntermediateEvent|EndEvent| ↩
                                        |Task|Sub-Process|Gateway)
 if ($fo/Categories@IsP3PPurpose)
  then return
   some $purposes in p3p:POLICIES/p3p:POLICY//p3p:PURPOSES
   satisfies $purposes = $fo/Categories
  else return fn:false()
```

Listing 6.9: The XPath version of the Purpose algorithm

```
if (//MessageFlow/Target[@P3PRecipient])
 then some $mf in p3p:POLICIES/p3p:POLICY/p3p:STATEMENT/p3p:RECIPIENT
  satisfies $mf = //MessageFlow/Target@P3PRecipient
```

Listing 6.10: The XPath version of the Recipient algorithm

6.4 Graphical Representation Inside the Model

We provide an easy way to aid users to identify privacy policies adherence. We extend the graphical layout of BPMN elements adding three simple markers over every element involved in a privacy policy. The three markers are shown in Fig. 6.5 and represent the complete adherence of an element to its relative privacy policy constraints, a warning message in case of a partial constraints compliance and an alert message if the element does not meet the privacy policy requirements.

Figure 6.5: From the left: complete, partial or missing privacy policies compliance

The use of these signs is coded inside the XML linearization using a simple string attribute for every element affected from the P3P code. In List. 6.11 it is possible to see an example for Tasks.

```
<TASK ID=‘‘T001’’ Name=‘‘Receive request’’ IsP3PCompliant=‘‘Warning’’/>
<TASK ID=‘‘T002’’ Name=‘‘Display form page’’ IsP3PCompliant=‘‘Yes’’/>
<TASK ID=‘‘T003’’ Name=‘‘Receive query’’ IsP3PCompliant=‘‘Warning’’/>
<TASK ID=‘‘T004’’ Name=‘‘Compute query’’ IsP3PCompliant=‘‘Yes’’/>
<TASK ID=‘‘T005’’ Name=‘‘Display results page’’ IsP3PCompliant=‘‘Yes’’/>
```

Listing 6.11: An example of signs coded inside the BPeX XML linearization

Using the motivating example, we assume that the search engine changes its privacy policy but not the process. The new provided privacy policy denies the search engine to collect any navigation information for further analysis. As it is possible to see in Fig. 6.6 – a simplified version of the running example – the tasks Receive Request and Receive Query are marked with a warning sign, because they could collect some not allowed data during the navigation and querying activities.

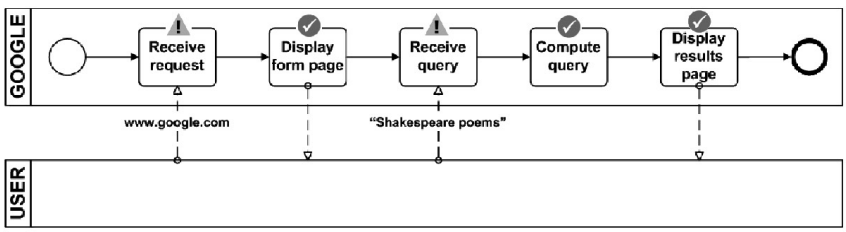


Figure 6.6: The example process marked with the new signs

6.5 Some Concluding Remarks

In this chapter, we extended BPeX notation with the support for P3P policies. We showed the feasibility to query the BPeX representation of a BPD extended with P3P statements, in order to verify its adherence to a given P3P privacy policy specification and a graphical way to help users to find errors or partial adherence of a process with respect to its privacy policy.

We aim to extend the graphical notation we presented in Sect. 6.4 in order to label any relevant element of a Process. The P3P developing process has stopped in 2006, and so we argue to apply these results using other technologies still supported or under development. We have shown a core integration between a BPMN serialization and some of the P3P XML-Schema elements. This integration could be extended to comprehend all the P3P elements. Also the BPeX linearization could be changed once a standard BPMN linearization will be published.

The core of the work presented in this section has been discussed at WOSIS 2008 Workshop [6] while the whole chapter will be soon published (early 2009) inside the JRPIT (Journal of Research and Practice in Information Technology) journal, published by the Australian Computer Society Inc.

Chapter 7

Conclusions and Further Works

With this work we have proposed several original contributions to business process modeling. There yet exist some proposals for a new BPMN conceptual model, and we have presented in this thesis our independent proposal. It is developed from scratch, in order to address the weak points of the other proposals and at the same time to suggest some advantageous alternatives. As said we have proposed an alternative conceptual model for BPMN and an XML serialization of such model called BPeX; we have developed a business process design methodology introducing an early definition of business process normal form for processes which satisfy some properties; we have added to business process modeling some new features like business processes views and user access management. Following the same structure we used in Section 1.3, we summarize the achieved results we have discussed in this work.

1. **Conceptual Model for BPMN.** There were different proposals to equip BPMN with a conceptual model, but the solution currently adopted remains unclear and too complex. We developed our model starting from scratch, and the result of this work is a complete conceptual model for BPMN with a plain metamodel and its related XML-based serialization. We called the conceptual model BPeX and we have detailed it in Chap. 2.

2. **Comparison with Other Standards.** We have provided in Chap. 3 a comparison between BPeX and BPEL or XPDL, as they are the most spread linearization proposals for BPMN. We have shown how BPeX can overcome the limitations of BPEL and XPDL using a running example.

3. **Business Processes Design Methodology.** We have provided in Chap. 4 a business process diagrams (BPD) design methodology and the related concept of business processes normal form (BPNF).

4. **Business Processes Diagrams Views.** BPMN does not provide natively a way to represent final users interactions with business process diagrams.

In Chap. 5 we have outlined the concept of business process diagram views (BPDV), a new mechanism to represent BPD elements subsets which can create through the application of certain requirements and properties. We have also handled the problem of views update fixing some conditions by which business process views can be updatable.

5. **Business Process Security Aspects.** In Section 5.4 we have outlined the users permissions management to access business processes views. We have discussed in the Chap. 6 another successfully example integrating BPMN with Web-oriented privacy policies.

Once the forthcoming new BPMN standard version will be published by OMG we plan to improve some concepts we outlined in this thesis, such as the design methodology, business processes views and the related security aspects. All these improvements will be reflected also in the BPeX XML-Schema model.

There are some directions we aim to follow starting from the results achieved during the last three years and summarized in this work. In the following list we want to summarize some future works in no particular order:

- Execution capabilities. BPMN 1.1 has not a native execution environment but leans on BPEL. We aim to extend the forthcoming version of BPMN with execution capabilities, borrowing from the Service-Oriented Architecture the needed methods.

- Metrics applications. One of the most claimed feature for BPMN is the capability to add values to diagrams elements according to whatever specified metric (e.g., time and costs metrics), computing values which could be analyzed to make the process more efficient.

- Modeling and validation support tools. Users usually dislike to model a business process by writing XML code or positioning elements without predefined procedures and hints. We aim to continue the develop of our graphical editor for BPMN in order to provide a tool which forces the modeler to use the correct element, giving back some feedbacks in the case of bad-formed diagrams.

- Companies constraints automatic checking. It would be very interesting to add guidance to BPMN to support users on modeling business processes considering international or companies inbound standards procedures.

- XML-based query language for business processes. Once a standard linearization for BPMN will have been published, we aim to adapt the queries we currently use in BPeX also to BPMN, and extending also the set of available interrogations, providing a complete query language for BPMN.

Bibliography

[1] Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. An XPath-based Preference Language for P3P. In *12th International World Wide Web Conference* (May 2003).

[2] Awad, A. Bpmn-q: A language to query business processes. In *EMISA* (2007), M. Reichert, S. Strecker, and K. Turowski, Eds., vol. P-119 of *LNI*, GI, pp. 115–128.

[3] Awad, A., Decker, G., and Weske, M. Efficient compliance checking using bpmn-q and temporal logic. In Dumas et al. [16], pp. 326–341.

[4] Bertino, E., Crampton, J., and Paci, F. Access Control and Authorization Constraints for WS-BPEL. *icws 0* (2006), 275–284.

[5] Business Rules Group. The Business Motivation Model - Business Governance in a Volatile World, Release 1.2. Tech. rep., Business Rules Group, 2005.

[6] Chinosi, M., and Trombetta, A. Integrating privacy policies into business processes. In *WOSIS* (2008), A. Rodríguez, M. I. Y. del Valle, and E. Fernández-Medina, Eds., INSTICC Press, pp. 13–25.

[7] Cranor, L., Dobbs, B., Hogben, G., Marchiori, M., Schunter, M., et al. The Platform for Privacy Preferences 1.1 (P3P1.1) Specification, November 2006. http://www.w3.org/TR/P3P11/.

[8] Curtis, W., and Alden, J. The Business Process Maturity Model (BPMM): What, Why and How. *BPM and Organisational Maturity, BP Trends Online Journal, Available 5* (2007), 02–07.

[9] Davenport, T. H. *Process Innovation – Reengineering Work through Information Technology.* Harvard Business School Press, 1993.

[10] Davenport, T. H., and Short, J. E. The new industrial engineering: Information technology and business process redesign. *Sloan Management Review 31*, 4 (1990), 11–27.

[11] Decker, G., Dijkman, R. M., Dumas, M., and García-Bañuelos, L. Transforming bpmn diagrams into yawl nets. In Dumas et al. [16], pp. 386–389.

[12] Decker, G., and Mendling, J. Instantiation semantics for process models. In Dumas et al. [16], pp. 164–179.

[13] Dijkman, R. M. Choreography-Based Design of Business Collaborations. Tech. rep., Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, 2006.

[14] Dijkman, R. M. A Classification of Differences between Similar Business Processes. In *Proceedings of the 11th IEEE EDOC Conference (EDOC)* (2007), pp. 37–47.

[15] Dijkman, R. M., Dumas, M., and Ouyang, C. Semantics and Analysis of Business Process Models in BPMN. *Information and Software Technology In Press, Corrected Proof* (2008).

[16] Dumas, M., Reichert, M., and Shan, M.-C., Eds. *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings* (2008), vol. 5240 of *Lecture Notes in Computer Science*, Springer.

[17] Gschwind, T., Koehler, J., and Wong, J. Applying patterns during business process modeling. In Dumas et al. [16], pp. 4–19.

[18] Hornung, T., Koschmider, A., and Mendling, J. Integration of heterogeneous BPM Schemas: The Case of XPDL and BPEL. In *CAiSE Forum* (2006), N. Boudjlida, D. Cheng, and N. Guelfi, Eds., vol. 231 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[19] Hoyer, V., Bucherer, E., and Schnabel, F. Collaborative e-business process modelling: Transforming private epc to public bpmn business process models. In ter Hofstede et al. [49], pp. 185–196.

[20] Karjoth, G., and Schunter, M. A Privacy Policy Model for Enterprises. In *CSFW* (2002), IEEE Computer Society, pp. 271–281.

[21] Korherr, B., and List, B. Extending the epc and the bpmn with business process goals and performance measures. In *ICEIS (3)* (2007), J. Cardoso, J. Cordeiro, and J. Filipe, Eds., pp. 287–294.

[22] Lee, J., Lee, D., and Kang, S. An Overview of the Business Process Maturity Model (BPMM). *LECTURE NOTES IN COMPUTER SCIENCE 4537* (2007), 384.

[23] Leymann, F., Roller, D., and Schmidt, M.-T. Web services and business process management. *IBM Systems Journal 41*, 2 (2002).

[24] Li, Y. H., Paik, H.-Y., Benatallah, B., and Benbernou, S. Formal Consistency Verification between BPEL process and Privacy Policy. In *Privacy Security Trust 2006 (PST 2006)* (Nov 2006), McGraw Hill, pp. 212–223.

[25] Malhotra, Y. Business Process Redesign: An Overview. *IEEE ENGINEERING MANAGEMENT REVIEW 26* (1998), 27–31.

[26] Mendling, J. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Institute of Information Systems and New Media Vienna University of Economics and Business Administration (WU Wien), May 2007.

[27] Mendling, J., de Laborda, C. P., and Zdun, U. Towards an Integrated BPM Schema: Control Flow Heterogeneity of PNML and BPEL4WS. In *Wissensmanagement (LNCS Volume)* (2005), K.-D. Althoff, A. Dengel, R. Bergmann, M. Nick, and T. Roth-Berghofer, Eds., vol. 3782 of *Lecture Notes in Computer Science*, Springer, pp. 570–579.

[28] Mendling, J., de Laborda, C. P., and Zdun, U. Towards Semantic Integration of XML-based Business Process Models. In *Wissensmanagement* (2005), K.-D. Althoff, A. Dengel, R. Bergmann, M. Nick, and T. Roth-Berghofer, Eds., DFKI, Kaiserslautern, pp. 513–517.

[29] Mendling, J., Neumann, G., and Nüttgens, M. A Comparison of XML Interchange Formats for Business Process Modelling. In *EMISA* (2004), F. Feltz, A. Oberweis, and B. Otjacques, Eds., vol. 56 of *LNI*, GI, pp. 129–140.

[30] Mendling, J., and Nüttgens, M. Epc syntax validation with xml schema languages. In *EPK* (2003), M. Nüttgens and F. J. Rump, Eds., GI-Arbeitskreis Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, pp. 19–30.

[31] Mendling, J., Strembeck, M., Stermsek, G., and Neumann, G. An Approach to Extract RBAC Models from BPEL4WS Processes. In *WETICE* (2004), IEEE Computer Society, pp. 81–86.

[32] O'Neil, P. E., and O'Neil, E. J. *Database: Principles, Programming, and Performance, Second Edition*. Morgan Kaufmann, 2000.

[33] Ouyang, C., van der Aalst, W. M., Dumas, M., and ter Hofstede, A. H. M. From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way. Tech. Rep. BPM-06-27, BPM Center, 2006. http://www.bpmcenter.org.

[34] Ouyang, C., van der Aalst, W. M., Dumas, M., and ter Hofstede, A. H. M. Translating BPMN to BPEL. Tech. Rep. BPM-06-02, BPM Center, 2006. http://www.bpmcenter.org.

[35] Polyvyanyy, A., and Weske, M. Hypergraph-based Modeling of Ad-Hoc Business Processes. In *Proceedings of the 1st International Workshop on Process Management for Highly Dynamic and Pervasive Scenarios (BPM: PM4HDPS)* (Milan, Italy, 0 2008).

[36] Recker, J., Indulska, M., Rosemann, M., and Green, P. Do Process Modelling Techniques Get Better? A Comparative Ontological Analysis of BPMN. In *Proceedings 16th Australasian Conference on Information Systems* (2005), Campbell, Bruce and Underwood, Jim and Bunker, Deborah, Eds.

[37] Recker, J., and Mendling, J. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. In *Proceedings 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortiums* (2006), Latour, Thibaud and Petit, Michael, Eds., pp. 521–532.

[38] Recker, J., Wohed, P., and Rosemann, M. Representation theory versus workflow patterns - the case of bpmn. In *ER* (2006), D. W. Embley, A. Olivé, and S. Ram, Eds., vol. 4215 of *Lecture Notes in Computer Science*, Springer, pp. 68–83.

[39] Reijers, H. A., and Mendling, J. Modularity in process models: Review and effects. In Dumas et al. [16], pp. 20–35.

[40] Rosemann, M. Understanding and impacting the practice of business process management. In Dumas et al. [16], p. 2.

[41] Rosemann, M., Recker, J., Indulska, M., and Green, P. Evaluating the Emerging Process Modelling Standard: Insights from Theory and Practice. In *Proceedings of the 14th European Conference on Information Systems (ECIS 2006)* (2006).

[42] Sadiq, W., and Orlowska, M. E. Applying graph reduction techniques for identifying structural conflicts in process models. In *CAiSE* (1999), M. Jarke and A. Oberweis, Eds., vol. 1626 of *Lecture Notes in Computer Science*, Springer, pp. 195–209.

[43] Sadiq, W., and Orlowska, M. E. Analyzing process models using graph reduction techniques. *Inf. Syst. 25*, 2 (2000), 117–134.

[44] Shudi, G., Sperberg-McQueen, C. M., and Thompson, H. S. W3C XML Schema Definition Language (XSD) 1.1 part 1. http://www.w3.org/TR/xmlschema11-1/, June 2008.

[45] Silver, B. Bpms watch. Internet Web-site, December 2007. http://www.brsilver.com/wordpress/.

[46] Swenson, K. The BPMN-XPDL-BPEL value chain. In blog "Go Flow", May 2006. http://kswenson.wordpress.com/2006/05/26/bpmn-xpdl-and-bpel/.

[47] Swenson, K. Go flow. Internet Web-site, December 2007. http://kswenson.wordpress.com/.

[48] Swenson, K. D., and Shapiro, R. M. BPM in Practice. www.lulu.com/content/2244958, 2008.

[49] ter Hofstede, A. H. M., Benatallah, B., and Paik, H.-Y., Eds. *Business Process Management Workshops, BPM 2007 International Workshops, BPI, BPD, CBP, ProHealth, RefMod, semantics4ws, Brisbane, Australia, September 24, 2007, Revised Selected Papers* (2008), vol. 4928 of *Lecture Notes in Computer Science*, Springer.

[50] The Object Management Group. Business Process Definition Metamodel (BPDM). Internet Web-site, June 2007. http://modeldriven.org/c/portal/layout?p_l_id=PUB.1040.1.

[51] The Object Management Group. Semantics of Business Vocabulary and Business Rules. Tech. rep., The Object Management Group, 2008.

[52] van der Aalst, W. M. Pi Calculus Versus Petri Nets: Let us eat "humble pie" rather than further inflate the "Pi hype", 2003. http://is.tm.tue.nl/staff/wvdaalst/publications/pi-hype.pdf.

[53] van der Aalst, W. M., and Lassen, K. B. Translating Workflow Nets to BPEL4WS. Tech. Rep. BPM-05-16, BPM Center, 2005. http://www.bpmcenter.org.

[54] van der Aalst, W. M., ter Hofstede, A. H. M., and Weske, M. Business Process Management: A Survey. In *Business Process Management* (2003), W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, Eds., vol. 2678 of *Lecture Notes in Computer Science*, Springer, pp. 1–12.

[55] van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. Workflow patterns. *Distributed and Parallel Databases 14*, 1 (2003), 5–51.

[56] van Dongen, B., Dijkman, R. M., and Mendling, J. Measuring Similarity between Business Process Models. Tech. rep., Eindhoven University of Technology, Eindhoven, The Netherlands, 2007.

[57] van Dongen, B. F., van der Aalst, W. M. P., and Verbeek, H. M. W. Verification of epcs: Using reduction rules and petri nets. In *CAiSE* (2005), O. Pastor and J. F. e Cunha, Eds., vol. 3520 of *Lecture Notes in Computer Science*, Springer, pp. 372–386.

[58] Vanhatalo, J., Völzer, H., and Koehler, J. The refined process structure tree. In Dumas et al. [16], pp. 100–115.

[59] WfMC. Workflow Management Coalition - Terminology & Glossary. On WfMC website, Feb 1999. WFMC-TC-1011. http://www.wfmc.org/standards/docs.htm.

[60] WfMC. Process Definition Interface – XML Process Definition Language. WfMC website, October 2005. WFMC-TC-1025. http://www.wfmc.org/standards/docs.htm.

[61] White, S. A. Process Modeling Notations and Workflow Patterns. On BPMN website, 2004. http://www.bpmn.org.

[62] White, S. A. Business Process Modeling Notation - OMG Final Adopted Specification. On BPMN website, 2006. http://www.bpmn.org.

[63] White, S. A. Business Process Modeling Notation, V1.1 - OMG Available Specification. On BPMN website, 2008. http://www.bpmn.org.

[64] Wohed, P., van der Aalst, W. M., Dumas, M., and ter Hofstede, A. H. M. *Analysis of Web Services Composition Languages: The Case of BPEL4WS*, vol. 2813/2003 of *LNCS*. Springer Berlin / Heidelberg, 2003.

[65] Wohed, P., van der Aalst, W. M., Dumas, M., ter Hofstede, A. H. M., and Russell, N. Pattern-based Analysis of BPMN - An extensive evaluation of the Control-flow, the Data and the Resource Perspectives. Tech. Rep. BPM-06-17, BPM Center, 2006. http://www.bpmncenter.org.

[66] Zha, H., Yang, Y., Wang, J., and Wen, L. Transforming xpdl to petri nets. In ter Hofstede et al. [49], pp. 197–207.

[67] zur Muehlen, M. Getting Started With Business Process Modeling. IIR BPM Conference, Orlando, Florida, May 2008.

[68] zur Muehlen, M., and Recker, J. How much language is enough? theoretical and practical use of the business process modeling notation. In *CAiSE* (2008), Z. Bellahsene and M. Léonard, Eds., vol. 5074 of *Lecture Notes in Computer Science*, Springer, pp. 465–479.

Appendix A

BPeX XML Complete Code

Listing A.1: The BPeX XML complete code

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:bpex="http://bpex.sf.net/bpex-1.1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://bpex.sf.net/bpex-1.1"
  elementFormDefault="qualified" attributeFormDefault="qualified"
  version="1.1.2" xml:lang="en">

  <!-- ==== Some general information ==== -->

  <xs:annotation>
    <xs:documentation> This is the new version of BPeX XML-Schema file for
      BPMN 1.1 </xs:documentation>
    <xs:documentation source="http://bpex.sf.net/bpex-1.1/"> Here you can
      find the changelog and the roadmap for this project
    </xs:documentation>
  </xs:annotation>

  <!-- ==== Elements Definitions ==== -->

  <!-- ### BUSINESS PROCESS DIAGRAM ### -->

  <xs:element name="BPD" type="bpex:BPDType"/>

  <!-- ==== Complex Types Definitions ==== -->

  <!-- ### Supporting Types ### -->
  <xs:complexType name="BPMNElementType">
    <xs:sequence>
      <xs:element name="Category" type="bpex:CategoryType" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="Documentation" type="xs:string" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="Id" type="bpex:Object" use="required"/>
  </xs:complexType>

  <xs:complexType name="CategoryType">
    <xs:complexContent>
      <xs:extension base="bpex:BPMNElementType">
        <xs:sequence>
          <xs:element name="Name" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="ExpressionType">
    <xs:complexContent>
      <xs:extension base="bpex:BPMNElementType">
        <xs:sequence>
          <xs:element name="ExpressionBody" type="xs:string"/>
          <xs:element name="ExpressionLanguage" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="PropertyType">
    <xs:complexContent>
      <xs:extension base="bpex:BPMNElementType">
        <xs:sequence>
          <xs:element name="Value" type="bpex:ExpressionType"
            minOccurs="0"/>
        </xs:sequence>
        <xs:attributeGroup ref="bpex:PropertyAttributes"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="AssignmentType">
```

```
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:sequence>
        <xs:element name="To" type="bpex:PropertyType"/>
        <xs:element name="From" type="bpex:ExpressionType"/>
      </xs:sequence>
      <xs:attribute name="AssignTime" default="Start">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Start"/>
            <xs:enumeration value="End"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ArtifactInputType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:attribute name="ArtifactRef" type="bpex:ObjectRef"
        use="required"/>
      <xs:attribute name="RequiredForStart" type="xs:boolean" default="1"
      />
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ArtifactOutputType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:attribute name="ArtifactRef" type="bpex:ObjectRef"
        use="required"/>
      <xs:attribute name="ProduceAtCompletion" type="xs:boolean"
        default="1"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="EntityType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:attribute name="Name" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="MessageType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:sequence>
        <xs:element name="Properties" type="bpex:PropertyType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="Name" type="xs:string" use="required"/>
      <xs:attribute name="FromRef" type="bpex:ObjectRef" use="required"/>
      <xs:attribute name="ToRef" type="bpex:ObjectRef" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="InputSetType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:sequence>
        <xs:element name="ArtifactInputs" type="bpex:ArtifactInputType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="PropertyInputs" type="bpex:PropertyType"
          minOccurs="0" maxOccurs="unbounded"/>
```

```
      </xs:sequence>                                                          140
    </xs:extension>                                                           141
  </xs:complexContent>                                                        142
</xs:complexType>                                                             143
                                                                              144
<xs:complexType name="OutputSetType">                                         145
  <xs:complexContent>                                                         146
    <xs:extension base="bpex:BPMNElementType">                                147
      <xs:sequence>                                                           148
        <xs:element name="ArtifactOutputs" type="bpex:ArtifactOutputType"     149
          minOccurs="0" maxOccurs="unbounded"/>                               150
        <xs:element name="PropertyOutputs" type="bpex:PropertyType"           151
          minOccurs="0" maxOccurs="unbounded"/>                               152
      </xs:sequence>                                                          153
    </xs:extension>                                                           154
  </xs:complexContent>                                                        155
</xs:complexType>                                                             156
                                                                              157
<xs:complexType name="ParticipantType">                                       158
  <xs:complexContent>                                                         159
    <xs:extension base="bpex:BPMNElementType">                                160
      <xs:sequence>                                                           161
        <xs:element name="Role" type="bpex:RoleType" minOccurs="0"/>          162
        <xs:element name="Entity" type="bpex:EntityType" minOccurs="0"/>      163
      </xs:sequence>                                                          164
      <xs:attribute name="ParticipantType" default="Role">                    165
        <xs:simpleType>                                                       166
          <xs:restriction base="xs:string">                                   167
            <xs:enumeration value="Role"/>                                    168
            <xs:enumeration value="Entity"/>                                  169
          </xs:restriction>                                                   170
        </xs:simpleType>                                                      171
      </xs:attribute>                                                         172
    </xs:extension>                                                           173
  </xs:complexContent>                                                        174
</xs:complexType>                                                             175
                                                                              176
<xs:complexType name="RoleType">                                              177
  <xs:complexContent>                                                         178
    <xs:extension base="bpex:BPMNElementType">                                179
      <xs:attribute name="Name" type="xs:string" use="required"/>             180
    </xs:extension>                                                           181
  </xs:complexContent>                                                        182
</xs:complexType>                                                             183
                                                                              184
<xs:complexType name="CommonSwimlaneType">                                    185
  <xs:complexContent>                                                         186
    <xs:extension base="bpex:BPMNElementType">                                187
      <xs:attribute name="Name" type="xs:string" use="required"/>             188
    </xs:extension>                                                           189
  </xs:complexContent>                                                        190
</xs:complexType>                                                             191
                                                                              192
<xs:complexType name="CommonConnectingObjectType">                            193
  <xs:complexContent>                                                         194
    <xs:extension base="bpex:BPMNElementType">                                195
      <xs:attribute name="Name" type="xs:string" use="required"/>             196
      <xs:attribute name="SourceRef" type="bpex:ObjectRef" use="required"/>   197
      <xs:attribute name="TargetRef" type="bpex:ObjectRef" use="required"/>   198
    </xs:extension>                                                           199
  </xs:complexContent>                                                        200
</xs:complexType>                                                             201
                                                                              202
<xs:complexType name="CommonFlowObjectType">                                  203
  <xs:complexContent>                                                         204
    <xs:extension base="bpex:BPMNElementType">                                205
      <xs:sequence>                                                           206
        <xs:element name="Assignments" type="bpex:AssignmentType"             207
          minOccurs="0" maxOccurs="unbounded"/>                               208
      </xs:sequence>                                                          209
```

```
      <xs:attribute name="Name" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ConditionType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:sequence>
        <xs:element name="ConditionExpression" type="bpex:ExpressionType"
          minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="Name" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="TimeDateExpressionType">
  <xs:complexContent>
    <xs:extension base="bpex:ExpressionType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="SignalType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:sequence>
        <xs:element name="Properties" type="bpex:PropertyType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="Name" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="CommonEventDetailType" abstract="true">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:attribute name="EventDetailType" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="None"/>
            <xs:enumeration value="Message"/>
            <xs:enumeration value="Timer"/>
            <xs:enumeration value="Error"/>
            <xs:enumeration value="Conditional"/>
            <xs:enumeration value="Link"/>
            <xs:enumeration value="Signal"/>
            <xs:enumeration value="Compensate"/>
            <xs:enumeration value="Cancel"/>
            <xs:enumeration value="Terminate"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ConditionalEventDetailType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonEventDetailType">
      <xs:sequence>
        <xs:element name="ConditionRef" type="bpex:ConditionType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="CompensationEventDetailType">
  <xs:complexContent>
```

```
    <xs:extension base="bpex:CommonEventDetailType">
      <xs:attribute name="ActivityRef" type="bpex:ObjectRef"
        use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ErrorEventDetailType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonEventDetailType">
      <xs:sequence>
        <xs:element name="ErrorCode" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="LinkEventDetailType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonEventDetailType">
      <xs:attribute name="Name" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="MessageEventDetailType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonEventDetailType">
      <xs:sequence>
        <xs:element name="WebService" type="bpex:WebServiceType"
          minOccurs="0"> </xs:element>
      </xs:sequence>
      <xs:attribute name="MessageRef" type="bpex:ObjectRef"
        use="required"/>
      <xs:attribute name="Implementation" default="Web␣Service">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Web␣Service"/>
            <xs:enumeration value="Other"/>
            <xs:enumeration value="Unspecified"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="SignalEventDetailType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonEventDetailType">
      <xs:sequence>
        <xs:element name="SignalRef" type="bpex:SignalType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="TimerEventDetailType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonEventDetailType">
      <xs:sequence>
        <xs:element name="TimeDate" type="bpex:TimeDateExpressionType"
          minOccurs="0"/>
        <xs:element name="TimeCycle" type="bpex:TimeDateExpressionType"
          minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="NoneEventDetailType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonEventDetailType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="CancelEventDetailType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonEventDetailType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="TerminateEventDetailType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonEventDetailType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="TransactionType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:attribute name="TransactionId" type="xs:string"/>
      <xs:attribute name="TransactionProtocol" type="xs:string"/>
      <xs:attribute name="TransactionMethod" default="Compensate">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Compensate"/>
            <xs:enumeration value="Store"/>
            <xs:enumeration value="Image"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="GateType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:sequence>
        <xs:element name="Assignments" type="bpex:AssignmentType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="OutgoingSequenceFlowRef" type="bpex:ObjectRef"
        use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="WebServiceType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:sequence>
        <xs:element name="Interface" type="xs:string"/>
        <xs:element name="Operation" type="xs:string"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="ParticipantRef" type="bpex:ObjectRef"
        use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ### BUSINESS PROCESS DIAGRAM ### -->

<xs:complexType name="BPDType">
  <xs:sequence>
    <xs:element name="Documentation" minOccurs="0" type="xs:string"/>
    <xs:element name="Pool" type="bpex:PoolType" maxOccurs="unbounded"
      minOccurs="1">
```

```
      <xs:annotation>
        <xs:documentation> Modification: BPMN specification requires this
          attribute to be a list of references to Pool IDs (pp.240-241).
          BPeX, instead, makes a strong hierarchical connection between
          BPDs and Pools inserting here Pools as children elements.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="DataObject" type="bpex:DataObjectType"
      minOccurs="0" maxOccurs="unbounded"> </xs:element>
    <xs:element name="Annotation" type="bpex:AnnotationType"
      minOccurs="0" maxOccurs="unbounded"> </xs:element>
    <xs:element name="Group" type="bpex:GroupType" minOccurs="0"
      maxOccurs="unbounded"> </xs:element>
    <xs:element name="MessageFlow" minOccurs="0" maxOccurs="unbounded"
      type="bpex:MessageFlowType"/>
    <xs:element name="Association" minOccurs="0" maxOccurs="unbounded"
      type="bpex:AssociationType"/>
  </xs:sequence>
  <xs:attributeGroup ref="bpex:BPDAttributes"/>
</xs:complexType>

<!-- ### PROCESS ### -->

<xs:complexType name="ProcessType">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:sequence>
        <xs:element name="GraphicalElements" type="bpex:ObjectRef"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Assignments" type="bpex:AssignmentType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Performers" type="xs:string" minOccurs="0"
          maxOccurs="unbounded"/>
        <xs:element name="Properties" type="bpex:PropertyType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="InputSets" type="bpex:InputSetType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="OutputSet" type="bpex:OutputSetType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="AdHocOrdering" type="bpex:AdHocOrderingType"
          default="Parallel" minOccurs="0"/>
        <xs:element name="AdHocCompletionCondition"
          type="bpex:ExpressionType" minOccurs="0"/>
      </xs:sequence>
      <xs:attributeGroup ref="bpex:ProcessAttributes"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ### SWIMLANES ### -->

<xs:complexType name="PoolType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonSwimlaneType">
      <xs:sequence>
        <xs:element name="Process" type="bpex:ProcessType"/>
        <xs:element name="Participant" type="bpex:ParticipantType"/>
        <xs:element name="Lane" maxOccurs="unbounded"
          type="bpex:LaneType"/>
        <xs:element name="SequenceFlow" minOccurs="0"
          maxOccurs="unbounded" type="bpex:SequenceFlowType"/>
      </xs:sequence>
      <xs:attributeGroup ref="bpex:PoolAttributes"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="LaneType">
  <xs:complexContent>
```

```
    <xs:extension base="bpex:CommonSwimlaneType">
      <xs:sequence>
        <xs:element name="StartEvent" type="bpex:StartEventType"
          minOccurs="0" maxOccurs="unbounded"> </xs:element>
        <xs:element name="IntermediateEvent"
          type="bpex:IntermediateEventType" minOccurs="0"
          maxOccurs="unbounded"> </xs:element>
        <xs:element name="EndEvent" type="bpex:EndEventType"
          minOccurs="0" maxOccurs="unbounded"> </xs:element>
        <xs:element name="Task" type="bpex:TaskType" minOccurs="0"
          maxOccurs="unbounded"> </xs:element>
        <xs:element name="Sub-Process" type="bpex:Sub-ProcessType"
          minOccurs="0" maxOccurs="unbounded"> </xs:element>
        <xs:element name="Gateway" type="bpex:CommonGatewayType"
          minOccurs="0" maxOccurs="unbounded"> </xs:element>
        <xs:element name="Lanes" type="bpex:ObjectRef" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ### EVENTS ### -->

<xs:complexType name="EventType" abstract="true">
  <xs:complexContent>
    <xs:extension base="bpex:CommonFlowObjectType">
      <xs:attribute name="EventType" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Start"/>
            <xs:enumeration value="End"/>
            <xs:enumeration value="Intermediate"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="StartEventType">
  <xs:complexContent>
    <xs:extension base="bpex:EventType">
      <xs:sequence>
        <xs:element name="Trigger" type="bpex:CommonEventDetailType"
          minOccurs="0" maxOccurs="unbounded"> </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="IntermediateEventType">
  <xs:complexContent>
    <xs:extension base="bpex:EventType">
      <xs:sequence>
        <xs:element name="Trigger" type="bpex:CommonEventDetailType"
          minOccurs="0" maxOccurs="unbounded"> </xs:element>
        <xs:element name="Target" type="bpex:ObjectRef" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="EndEventType">
  <xs:complexContent>
    <xs:extension base="bpex:EventType">
      <xs:sequence>
        <xs:element name="Result" type="bpex:CommonEventDetailType"
          minOccurs="0" maxOccurs="unbounded"> </xs:element>
      </xs:sequence>
```

```
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ### ACTIVITIES ### -->

<xs:complexType name="CommonActivityType" abstract="true">
  <xs:complexContent>
    <xs:extension base="bpex:CommonFlowObjectType">
      <xs:sequence>
        <xs:element name="Performers" type="xs:string" minOccurs="0"
          maxOccurs="unbounded"/>
        <xs:element name="Properties" type="bpex:PropertyType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="InputSets" type="bpex:InputSetType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="OutputSets" type="bpex:OutputSetType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="IORules" type="bpex:ExpressionType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attributeGroup ref="bpex:ActivityAttributes"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="StandardLoopActivityType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonActivityType">
      <xs:sequence>
        <xs:element name="LoopCondition" type="bpex:ExpressionType"/>
      </xs:sequence>
      <xs:attribute name="LoopCounter">
        <xs:simpleType>
          <xs:restriction base="xs:int">
            <xs:minInclusive value="0"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="LoopMaximum" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:int">
            <xs:minInclusive value="0"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="TestTime" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Before"/>
            <xs:enumeration value="After"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="MultiInstanceActivityType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonActivityType">
      <xs:sequence>
        <xs:element name="MI_Condition" type="bpex:ExpressionType"/>
        <xs:element name="ComplexMI_FlowCondition"
          type="bpex:ExpressionType" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="LoopCounter">
        <xs:simpleType>
          <xs:restriction base="xs:int">
            <xs:minInclusive value="0"/>
```

```
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="MI_Ordering" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Sequential"/>
            <xs:enumeration value="Parallel"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="MI_FlowCondition" default="All">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="None"/>
            <xs:enumeration value="One"/>
            <xs:enumeration value="All"/>
            <xs:enumeration value="Complex"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="TaskType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonActivityType">
      <xs:attribute name="TaskType" use="optional" default="None">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Service"/>
            <xs:enumeration value="Receive"/>
            <xs:enumeration value="Send"/>
            <xs:enumeration value="User"/>
            <xs:enumeration value="Script"/>
            <xs:enumeration value="Abstract"/>
            <xs:enumeration value="Manual"/>
            <xs:enumeration value="Reference"/>
            <xs:enumeration value="None"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="NoneTaskType">
  <xs:complexContent>
    <xs:extension base="bpex:TaskType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="AbstractTaskType">
  <xs:complexContent>
    <xs:extension base="bpex:TaskType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ServiceTaskType">
  <xs:complexContent>
    <xs:extension base="bpex:TaskType">
      <xs:sequence>
        <xs:element name="WebService" type="bpex:WebServiceType"
          minOccurs="0"> </xs:element>
      </xs:sequence>
      <xs:attribute name="InMessageRef" type="bpex:ObjectRef"
        use="required"/>
      <xs:attribute name="OutMessageRef" type="bpex:ObjectRef"
        use="required"/>
```

```
      <xs:attribute name="Implementation" default="Web␣Service">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Web␣Service"/>
            <xs:enumeration value="Other"/>
            <xs:enumeration value="Unspecified"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ReceiveTaskType">
  <xs:complexContent>
    <xs:extension base="bpex:TaskType">
      <xs:sequence>
        <xs:element name="WebService" type="bpex:WebServiceType"
          minOccurs="0"> </xs:element>
      </xs:sequence>
      <xs:attribute name="MessageRef" type="bpex:ObjectRef"
        use="required"/>
      <xs:attribute name="Instantiate" type="xs:boolean" default="0"/>
      <xs:attribute name="Implementation" default="Web␣Service">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Web␣Service"/>
            <xs:enumeration value="Other"/>
            <xs:enumeration value="Unspecified"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="SendTaskType">
  <xs:complexContent>
    <xs:extension base="bpex:TaskType">
      <xs:sequence>
        <xs:element name="WebService" type="bpex:WebServiceType"
          minOccurs="0"> </xs:element>
      </xs:sequence>
      <xs:attribute name="MessageRef" type="bpex:ObjectRef"
        use="required"/>
      <xs:attribute name="Implementation" default="Web␣Service">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Web␣Service"/>
            <xs:enumeration value="Other"/>
            <xs:enumeration value="Unspecified"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="UserTaskType">
  <xs:complexContent>
    <xs:extension base="bpex:TaskType">
      <xs:sequence>
        <xs:element name="WebService" type="bpex:WebServiceType"
          minOccurs="0"> </xs:element>
      </xs:sequence>
      <xs:attribute name="InMessageRef" type="bpex:ObjectRef"
        use="required"/>
      <xs:attribute name="OutMessageRef" type="bpex:ObjectRef"
        use="required"/>
      <xs:attribute name="Implementation" default="Web␣Service">
```

```
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Web Service"/>
            <xs:enumeration value="Other"/>
            <xs:enumeration value="Unspecified"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ScriptTaskType">
  <xs:complexContent>
    <xs:extension base="bpex:TaskType">
      <xs:sequence>
        <xs:element name="Script" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ManualTaskType">
  <xs:complexContent>
    <xs:extension base="bpex:TaskType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ReferenceTaskType">
  <xs:complexContent>
    <xs:extension base="bpex:TaskType">
      <xs:attribute name="TaskRef" type="bpex:ObjectRef" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="Sub-ProcessType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonActivityType">
      <xs:sequence>
        <xs:element name="Transaction" type="bpex:TransactionType"/>
      </xs:sequence>
      <xs:attribute name="SubProcessType" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Embedded"/>
            <xs:enumeration value="Reusable"/>
            <xs:enumeration value="Reference"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="IsATransaction" type="xs:boolean"
        use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="EmbeddedSubProcessType">
  <xs:complexContent>
    <xs:extension base="bpex:Sub-ProcessType">
      <xs:sequence>
        <xs:element name="GraphicalElements" type="bpex:ObjectRef"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="AdHocCompletionCondition"
          type="bpex:ExpressionType" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="AdHoc" type="xs:boolean" use="required"/>
      <xs:attribute name="AdHocOrdering" use="optional"
        default="Parallel">
        <xs:simpleType>
```

```
          <xs:restriction base="xs:string">
            <xs:enumeration value="Parallel"/>
            <xs:enumeration value="Sequential"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ReusableSubProcessType">
  <xs:complexContent>
    <xs:extension base="bpex:Sub-ProcessType">
      <xs:sequence>
        <xs:element name="InputMaps" type="bpex:ExpressionType"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="OutputMaps" type="bpex:ExpressionType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="DiagramRef" type="bpex:ObjectRef"
        use="required"/>
      <xs:attribute name="ProcessRef" type="bpex:ObjectRef"
        use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ReferenceSubProcessType">
  <xs:complexContent>
    <xs:extension base="bpex:Sub-ProcessType">
      <xs:attribute name="SubProcessRef" type="bpex:ObjectRef"
        use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ### GATEWAYS ### -->

<xs:complexType name="CommonGatewayType" abstract="true">
  <xs:complexContent>
    <xs:extension base="bpex:CommonFlowObjectType">
      <xs:sequence>
        <xs:element name="Gates" type="bpex:GateType" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="GatewayType" default="Data-Based␣Exclusive">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Data-Based␣Exclusive"/>
            <xs:enumeration value="Event-Based␣Exclusive"/>
            <xs:enumeration value="Inclusive"/>
            <xs:enumeration value="Complex"/>
            <xs:enumeration value="Parallel"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DataBasedExclusiveGatewayType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonGatewayType">
      <xs:attribute name="ExclusiveType" type="xs:string" fixed="Data"
        use="required"/>
      <xs:attribute name="MarkerVisible" type="xs:boolean" default="0"/>
      <xs:attribute name="DefaultGate" type="bpex:ObjectRef"
        use="optional"/>
    </xs:extension>
  </xs:complexContent>
```

```
</xs:complexType>

<xs:complexType name="EventBasedExclusiveGatewayType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonGatewayType">
      <xs:attribute name="ExclusiveType" type="xs:string" fixed="Event"
        use="required"/>
      <xs:attribute name="Instantiate" type="xs:boolean" default="0"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="InclusiveGatewayType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonGatewayType">
      <xs:attribute name="DefaultGate" type="bpex:ObjectRef"
        use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ComplexGatewayType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonGatewayType">
      <xs:sequence>
        <xs:element name="IncomingCondition" type="bpex:ExpressionType"
          minOccurs="0"/>
        <xs:element name="OutgoingCondition" type="bpex:ExpressionType"
          minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="ParallelGatewayType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonGatewayType"/>
  </xs:complexContent>
</xs:complexType>

<!-- ### ARTIFACTS ### -->

<xs:complexType name="ArtifactType" abstract="true">
  <xs:complexContent>
    <xs:extension base="bpex:BPMNElementType">
      <xs:attribute name="ArtifactType" default="DataObject">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="DataObject"/>
            <xs:enumeration value="Group"/>
            <xs:enumeration value="Annotation"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="DataObjectType">
  <xs:complexContent>
    <xs:extension base="bpex:ArtifactType">
      <xs:sequence>
        <xs:element name="Properties" type="bpex:PropertyType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="Name" type="xs:string" use="required"/>
      <xs:attribute name="State" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<xs:complexType name="GroupType">
  <xs:complexContent>
    <xs:extension base="bpex:ArtifactType">
      <xs:sequence>
        <xs:element name="GraphicalElements" type="bpex:ObjectRef"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="CategoryRef" type="bpex:ObjectRef"
        use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="AnnotationType">
  <xs:complexContent>
    <xs:extension base="bpex:ArtifactType">
      <xs:sequence>
        <xs:element name="Text" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- ### CONNECTING OBJECTS ###-->

<xs:complexType name="MessageFlowType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonConnectingObjectType">
      <xs:sequence>
        <xs:element name="Message" type="bpex:MessageType"
          minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="SequenceFlowType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonConnectingObjectType">
      <xs:sequence>
        <xs:element name="ConditionExpression" type="bpex:ExpressionType"
          minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="ConditionType" default="None">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="None"/>
            <xs:enumeration value="Expression"/>
            <xs:enumeration value="Default"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="AssociationType">
  <xs:complexContent>
    <xs:extension base="bpex:CommonConnectingObjectType">
      <xs:attribute name="Direction" default="None">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="None"/>
            <xs:enumeration value="One"/>
            <xs:enumeration value="Both"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
```

```
  </xs:complexContent>
</xs:complexType>

<!-- ==== Simple Types Definitions ==== -->

<xs:simpleType name="Object">
  <xs:restriction base="xs:ID"/>
</xs:simpleType>

<xs:simpleType name="ObjectRef">
  <xs:restriction base="xs:IDREF"/>
</xs:simpleType>

<xs:simpleType name="AdHocOrderingType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Parallel"/>
    <xs:enumeration value="Sequential"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="StatusType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="None"/>
    <xs:enumeration value="Ready"/>
    <xs:enumeration value="Active"/>
    <xs:enumeration value="Cancelled"/>
    <xs:enumeration value="Aborting"/>
    <xs:enumeration value="Aborted"/>
    <xs:enumeration value="Completing"/>
    <xs:enumeration value="Completed"/>
  </xs:restriction>
</xs:simpleType>

<!-- ==== Attribute Groups ==== -->

<xs:attributeGroup name="BPDAttributes">
  <xs:attribute name="Id" type="bpex:Object" use="required"/>
  <xs:attribute name="Name" type="xs:string" use="required"/>
  <xs:attribute name="Version" type="xs:string" use="optional"/>
  <xs:attribute name="Author" type="xs:string" use="optional"/>
  <xs:attribute name="Language" type="xs:string" default="English"
    use="optional"/>
  <xs:attribute name="QueryLanguage" type="xs:string" use="optional"/>
  <xs:attribute name="CreationDate" type="xs:dateTime" use="optional"/>
  <xs:attribute name="ModificationDate" type="xs:dateTime"
    use="optional"/>
</xs:attributeGroup>

<xs:attributeGroup name="ProcessAttributes">
  <xs:attribute name="Name" type="xs:string" use="required"/>
  <xs:attribute name="ProcessType" default="None">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="None"/>
        <xs:enumeration value="Private"/>
        <xs:enumeration value="Abstract"/>
        <xs:enumeration value="Collaboration"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="Status" default="None" type="bpex:StatusType"/>
  <xs:attribute name="AdHoc" type="xs:boolean" default="0"/>
</xs:attributeGroup>

<xs:attributeGroup name="PropertyAttributes">
  <xs:attribute name="Name" type="xs:string" use="required"/>
  <xs:attribute name="Type" type="xs:string" use="required"/>
  <xs:attribute name="Correlation" type="xs:boolean" default="0"/>
</xs:attributeGroup>
```

```
  <xs:attributeGroup name="PoolAttributes">                                    1120
    <xs:attribute name="ProcessRef" type="bpex:ObjectRef" use="optional"/>     1121
    <xs:attribute name="ParticipantRef" type="bpex:ObjectRef"                  1122
      use="required"/>                                                         1123
    <xs:attribute name="BoundaryVisible" type="xs:boolean" default="1"/>      1124
    <xs:attribute name="MainPool" type="xs:boolean" default="0"/>             1125
  </xs:attributeGroup>                                                         1126
                                                                               1127
  <xs:attributeGroup name="ActivityAttributes">                                1128
    <xs:attribute name="ActivityType" use="required">                          1129
      <xs:simpleType>                                                          1130
        <xs:restriction base="xs:string">                                      1131
          <xs:enumeration value="Task"/>                                       1132
          <xs:enumeration value="Sub-Process"/>                                1133
        </xs:restriction>                                                      1134
      </xs:simpleType>                                                         1135
    </xs:attribute>                                                            1136
    <xs:attribute name="Status" default="None" type="bpex:StatusType"/>        1137
    <xs:attribute name="StartQuantity" default="1">                            1138
      <xs:simpleType>                                                          1139
        <xs:restriction base="xs:int">                                         1140
          <xs:minInclusive value="1"/>                                         1141
        </xs:restriction>                                                      1142
      </xs:simpleType>                                                         1143
    </xs:attribute>                                                            1144
    <xs:attribute name="CompletionQuantity" default="1">                       1145
      <xs:simpleType>                                                          1146
        <xs:restriction base="xs:int">                                         1147
          <xs:minInclusive value="1"/>                                         1148
        </xs:restriction>                                                      1149
      </xs:simpleType>                                                         1150
    </xs:attribute>                                                            1151
    <xs:attribute name="LoopType" default="None">                              1152
      <xs:simpleType>                                                          1153
        <xs:restriction base="xs:string">                                      1154
          <xs:enumeration value="None"/>                                       1155
          <xs:enumeration value="Standard"/>                                   1156
          <xs:enumeration value="MultiInstance"/>                              1157
        </xs:restriction>                                                      1158
      </xs:simpleType>                                                         1159
    </xs:attribute>                                                            1160
  </xs:attributeGroup>                                                         1161
                                                                               1162
</xs:schema>                                                                   1163
```

Appendix B

Figures and XML Code

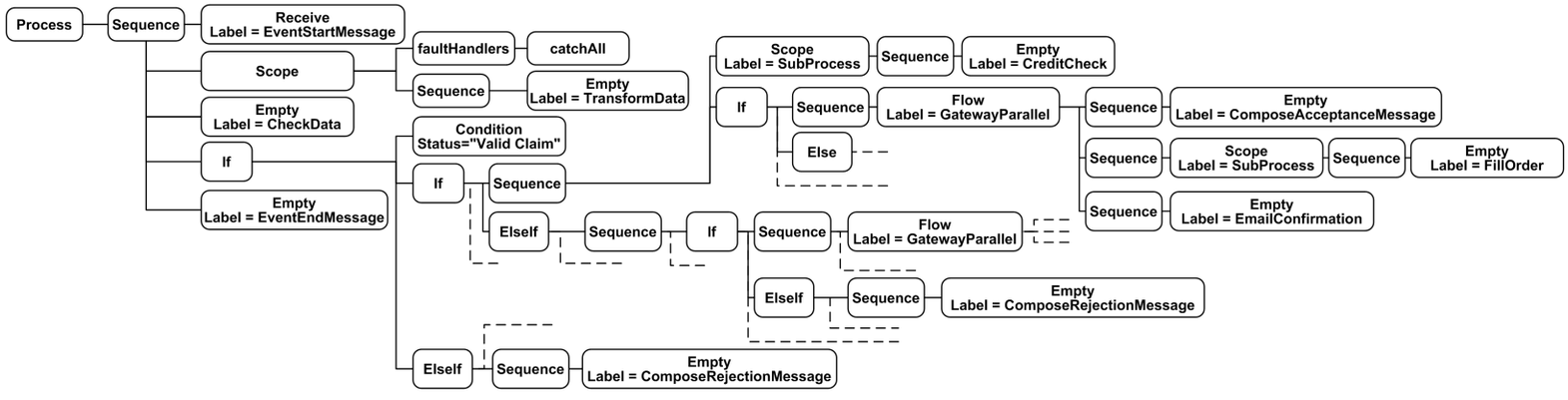


Figure B.1: A sketch of the graphical representation of the XML-tree structure for the BPEL serialization of the running example

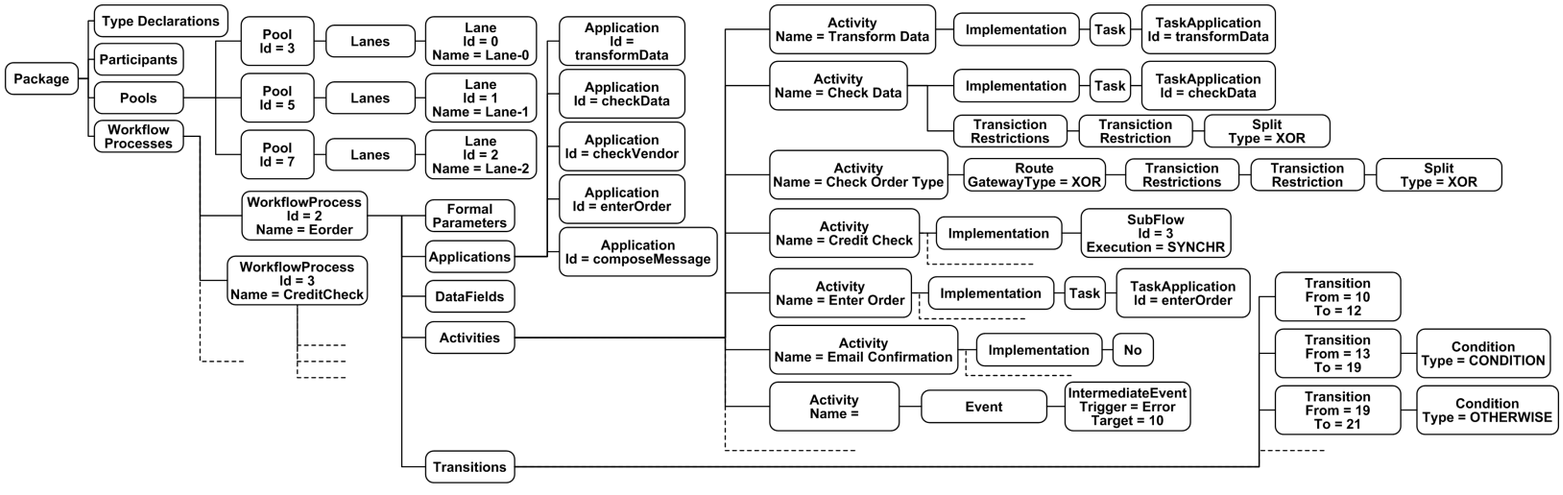


Figure B.2: The simplified version of the graphical layout of the XPDL tree model for the running example

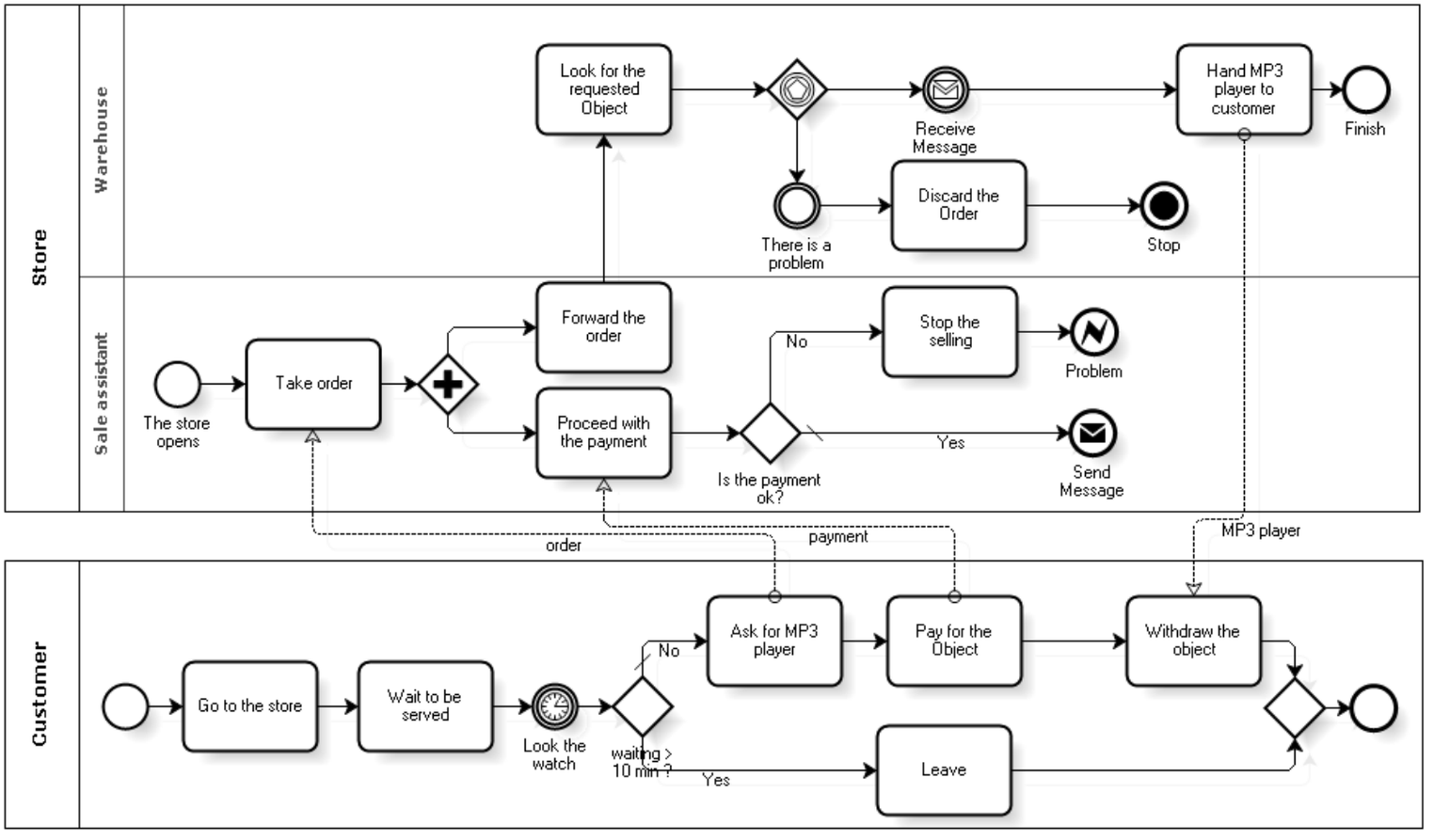


Figure B.3: The BPD at the end of the first phase of the methodology

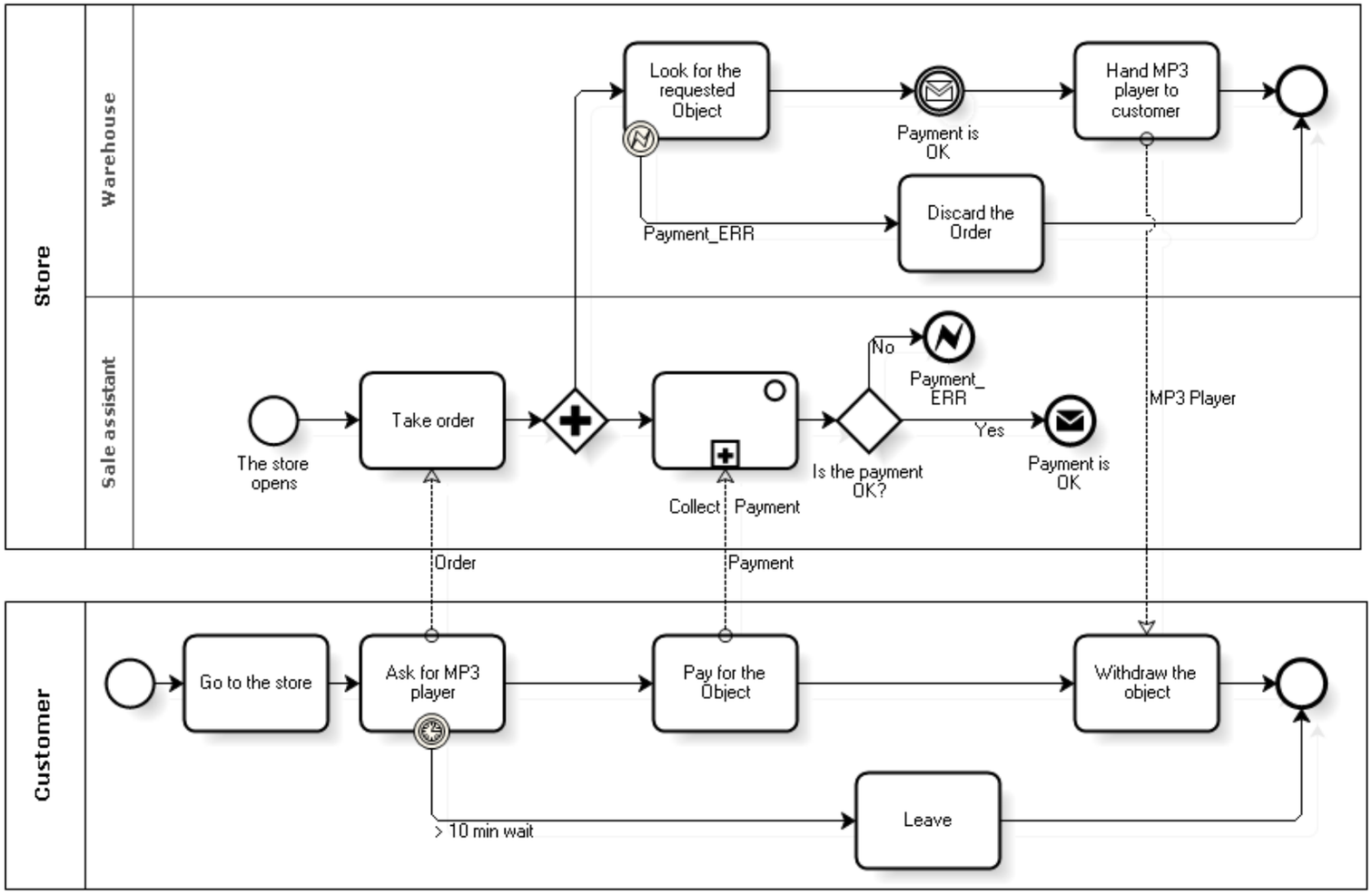


Figure B.4: The final version of the example process

Listing B.1: The BPeX XML code for the example used to present the business process design methodology

```
<?xml version="1.0" encoding="UTF-8"?>
<BPD xmlns="http://bpex.sf.net/bpex-1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://bpex.sf.net/bpex-1.1 bpex-1.1.2.xsd"
  xmlns:bpex="http://bpex.sf.net/bpex-1.1" bpex:Id="BPD_001"
  bpex:Name="Customer/Store Example">
  <Pool bpex:ParticipantRef="Part_01" bpex:Name="Store" bpex:Id="Pool_001">
    <Process bpex:Name="Store Process" bpex:Id="Proc_01"/>
    <Participant bpex:Id="Part_01"/>
    <Lane bpex:Name="Warehouseman" bpex:Id="Lane_001">
      <IntermediateEvent bpex:EventType="Intermediate"
        bpex:Name="Payment is OK" bpex:Id="IE_001">
        <Trigger bpex:EventDetailType="Message" bpex:Id="Tr_001"/>
      </IntermediateEvent>
      <IntermediateEvent bpex:EventType="Intermediate"
        bpex:Name="Payment_ERR" bpex:Id="IE_002">
        <Trigger bpex:EventDetailType="Error" bpex:Id="Tr_002"/>
        <Target>T_006</Target>
      </IntermediateEvent>
      <EndEvent bpex:EventType="End" bpex:Name="" bpex:Id="EE_001">
        <Result bpex:EventDetailType="None" bpex:Id="Res_001"/>
      </EndEvent>
      <Task bpex:ActivityType="Task"
        bpex:Name="Look for the requested Object" bpex:Id="T_006"/>
      <Task bpex:ActivityType="Task" bpex:Name="Discard the Order"
        bpex:Id="T_007"/>
      <Task bpex:ActivityType="Task"
        bpex:Name="Hand MP3 player to the customer" bpex:Id="T_008"/>
    </Lane>
    <Lane bpex:Name="Sale Assistant" bpex:Id="Lane_002">
      <StartEvent bpex:EventType="Start" bpex:Name="The store opens"
        bpex:Id="SE_001">
        <Trigger bpex:EventDetailType="None" bpex:Id="Tr_003"/>
      </StartEvent>
      <EndEvent bpex:EventType="End" bpex:Name="Payment_ERR"
        bpex:Id="EE_002">
        <Result bpex:EventDetailType="Error" bpex:Id="Res_002"/>
      </EndEvent>
      <EndEvent bpex:EventType="End" bpex:Name="Payment is OK"
        bpex:Id="EE_003">
        <Result bpex:EventDetailType="Message" bpex:Id="Res_003"/>
      </EndEvent>
      <Gateway bpex:Name="" bpex:Id="GW_001" bpex:GatewayType="Parallel">
        <Gates bpex:OutgoingSequenceFlowRef="SF_001"/>
        <Gates bpex:OutgoingSequenceFlowRef="SF_002"/>
      </Gateway>
      <Gateway bpex:Name="Is the payment OK?" bpex:Id="GW_002"
        bpex:GatewayType="Data-Based Exclusive">
        <Gates bpex:OutgoingSequenceFlowRef="SF_003"/>
        <Gates bpex:OutgoingSequenceFlowRef="SF_004"/>
      </Gateway>
      <Task bpex:ActivityType="Task" bpex:Name="Take order" bpex:Id="T_009"/>
      <Sub-Process bpex:SubProcessType="Embedded"
        bpex:ActivityType="Sub-Process" bpex:Name="Collect Payment"
        bpex:Id="SP_001"/>
    </Lane>
    <SequenceFlow bpex:Name="" bpex:SourceRef="GW_001"
      bpex:TargetRef="T_006" bpex:Id="SF_001"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="GW_001"
      bpex:TargetRef="SP_001" bpex:Id="SF_002"/>
    <SequenceFlow bpex:Name="No" bpex:SourceRef="GW_002"
      bpex:TargetRef="EE_002" bpex:Id="SF_003"/>
    <SequenceFlow bpex:Name="Yes" bpex:SourceRef="GW_002"
      bpex:TargetRef="EE_003" bpex:Id="SF_004" bpex:ConditionType="Default"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="SE_001"
      bpex:TargetRef="T_009" bpex:Id="SF_005"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_009"
```

```
      bpex:TargetRef="GW_001" bpex:Id="SF_005"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="SP_001"
      bpex:TargetRef="GW_002" bpex:Id="SF_006"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_006"
      bpex:TargetRef="IE_001" bpex:Id="SF_007"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="IE_001"
      bpex:TargetRef="T_008" bpex:Id="SF_008"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_008"
      bpex:TargetRef="EE_001" bpex:Id="SF_009"/>
    <SequenceFlow bpex:Name="Payment_ERR" bpex:SourceRef="IE_002"
      bpex:TargetRef="T_007" bpex:Id="SF_010"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_007"
      bpex:TargetRef="EE_001" bpex:Id="SF_011"/>
  </Pool>
  <Pool bpex:ParticipantRef="Part_02" bpex:Name="Customer"
    bpex:Id="Pool_002">
    <Process bpex:Name="Customer␣Process" bpex:Id="Proc_02"/>
    <Participant bpex:Id="Part_02"/>
    <Lane bpex:Name="Customer" bpex:Id="Lane_003">
      <StartEvent bpex:EventType="Start" bpex:Name="" bpex:Id="SE_002">
        <Trigger bpex:EventDetailType="None" bpex:Id="Tr_004"/>
      </StartEvent>
      <IntermediateEvent bpex:EventType="Intermediate" bpex:Name=">␣10␣min"
        bpex:Id="IE_003">
        <Trigger bpex:EventDetailType="Timer" bpex:Id="Tr_005"/>
        <Target>T_002</Target>
      </IntermediateEvent>
      <EndEvent bpex:EventType="End" bpex:Name="" bpex:Id="EE_004">
        <Result bpex:EventDetailType="None" bpex:Id="Res_004"/>
      </EndEvent>
      <Task bpex:ActivityType="Task" bpex:Name="Go␣to␣the␣Store"
        bpex:Id="T_001"/>
      <Task bpex:ActivityType="Task" bpex:Name="Ask␣for␣MP3␣Player"
        bpex:Id="T_002"/>
      <Task bpex:ActivityType="Task" bpex:Name="Leave" bpex:Id="T_003"/>
      <Task bpex:ActivityType="Task" bpex:Name="Pay␣for␣the␣Object"
        bpex:Id="T_004"/>
      <Task bpex:ActivityType="Task" bpex:Name="Withdraw␣the␣object"
        bpex:Id="T_005"/>
    </Lane>
    <SequenceFlow bpex:Name="" bpex:SourceRef="SE_002"
      bpex:TargetRef="T_001" bpex:Id="SF_012"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_001"
      bpex:TargetRef="T_002" bpex:Id="SF_013"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_002"
      bpex:TargetRef="T_004" bpex:Id="SF_014"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_004"
      bpex:TargetRef="T_005" bpex:Id="SF_015"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_005"
      bpex:TargetRef="EE_004" bpex:Id="SF_016"/>
    <SequenceFlow bpex:Name=">␣10␣min" bpex:SourceRef="IE_003"
      bpex:TargetRef="T_003" bpex:Id="SF_017"/>
    <SequenceFlow bpex:Name="" bpex:SourceRef="T_003"
      bpex:TargetRef="EE_004" bpex:Id="SF_018"/>
  </Pool>
  <MessageFlow bpex:Name="Order" bpex:SourceRef="T_002"
    bpex:TargetRef="T_009" bpex:Id="MF_001">
    <Message bpex:Name="Order" bpex:FromRef="Part_02" bpex:ToRef="Part_01"
      bpex:Id="MSG_001"/></MessageFlow>
  <MessageFlow bpex:Name="Payment" bpex:SourceRef="T_004"
    bpex:TargetRef="SP_001" bpex:Id="MF_002">
    <Message bpex:Name="Payment" bpex:FromRef="Part_02"
      bpex:ToRef="Part_01" bpex:Id="MSG_002"/></MessageFlow>
  <MessageFlow bpex:Name="MP3␣Player" bpex:SourceRef="T_008"
    bpex:TargetRef="T_005" bpex:Id="MF_003">
    <Message bpex:Name="MP3␣Player" bpex:FromRef="Part_01"
      bpex:ToRef="Part_02" bpex:Id="MSG_003"/></MessageFlow>
</BPD>
```

Appendix C

P3P supporting material

```
<BPD Name=‘‘BPeX Example’’>
 <POOL ID=‘‘P001’’ NAME=‘‘Google’’>
  <LANE ID=‘‘L001’’ NAME=‘‘Google’’>
   <STARTEVENT ID=‘‘E001’’ EventType=‘‘Start’’>
    <TRIGGER EVENTDETAILTYPE=‘‘None’’/>
   </STARTEVENT>
   <TASK ID=‘‘T001’’ Name=‘‘Receive request’’ TASKTYPE=‘‘None’’/>
   <TASK ID=‘‘T002’’ Name=‘‘Display form page’’ TASKTYPE=‘‘None’’/>
   <TASK ID=‘‘T003’’ Name=‘‘Receive query’’ TASKTYPE=‘‘None’’/>
   <TASK ID=‘‘T004’’ Name=‘‘Compute query’’ TASKTYPE=‘‘None’’/>
   <TASK ID=‘‘T005’’ Name=‘‘Display results page’’ TASKTYPE=‘‘None’’/>
   <ENDEVENT ID=‘‘E002’’ EventType=‘‘End’’>
    <RESULT EVENTDETAILTYPE=‘‘None’’/>
   </ENDEVENT>
  </LANE>
  <SEQUENCEFLOW ID=‘‘SF001’’ SOURCEREF=‘‘E001’’ TARGETREF=‘‘T001’’/>
  <SEQUENCEFLOW ID=‘‘SF002’’ SOURCEREF=‘‘T001’’ TARGETREF=‘‘T002’’/>
  <SEQUENCEFLOW ID=‘‘SF003’’ SOURCEREF=‘‘T002’’ TARGETREF=‘‘T003’’/>
  <SEQUENCEFLOW ID=‘‘SF004’’ SOURCEREF=‘‘T003’’ TARGETREF=‘‘T004’’/>
  <SEQUENCEFLOW ID=‘‘SF005’’ SOURCEREF=‘‘T004’’ TARGETREF=‘‘T005’’/>
  <SEQUENCEFLOW ID=‘‘SF006’’ SOURCEREF=‘‘T005’’ TARGETREF=‘‘E002’’/>
 </POOL>
 <POOL ID=‘‘P002’’ NAME=‘‘User’’/>
 <MESSAGEFLOW ID=‘‘MF001’’ SOURCEREF=‘‘P002’’ TARGETREF=‘‘T001’’>
  <MESSAGE NAME=‘‘www.google.com’’/>
 </MESSAGEFLOW>
 <MESSAGEFLOW ID=‘‘MF002’’ SOURCEREF=‘‘T002’’ TARGETREF=‘‘P002’’>
  <MESSAGE/>
 </MESSAGEFLOW>
 <MESSAGEFLOW ID=‘‘MF003’’ SOURCEREF=‘‘P002’’ TARGETREF=‘‘T003’’>
  <MESSAGE NAME=‘‘Shakespeare poems’’/>
 </MESSAGEFLOW>
 <MESSAGEFLOW ID=‘‘MF004’’ SOURCEREF=‘‘T005’’ TARGETREF=‘‘P002’’>
  <MESSAGE/>
 </MESSAGEFLOW>
</BPD>
```

Listing C.1: The BPeX linearization of the BPMN diagram

```
<POLICIES xmlns="http://www.w3.org/2002/01/P3Pv1">
 <POLICY name="Google␣Example␣Policy"
  discuri="http://www.google.com/privacypolicy.html"
  xml:lang="en">
  <ENTITY>
   <EXTENSION>
    <p3p11:data-group>
     <p3p11:datatype>
      <p3p11:business>
       <p3p11:orgname>Google Inc.</p3p11:orgname>
       <p3p11:contact-info>
        <p3p11:postal>
         <p3p11:street>1600 Amph.Parkway</p3p11:street>
         <p3p11:city>Mountain View</p3p11:city>
         <p3p11:state>CA</p3p11:state>
         <p3p11:postalcode>94043</p3p11:postalcode>
         <p3p11:country>USA</p3p11:country>
        </p3p11:postal>
       </p3p11:contact-info>
      </p3p11:business>
     </p3p11:datatype>
    </p3p11:data-group>
   </EXTENSION>
   <DATA-GROUP>
    <DATA ref="...">for backward compatibility</DATA>
   </DATA-GROUP>
  </ENTITY>
  <ACCESS><nonident/></ACCESS>
  <STATEMENT>
   <PURPOSE><admin/><develop/></PURPOSE>
   <RECIPIENT><ours/></RECIPIENT>
   <RETENTION><stated-purpose/></RETENTION>
   <DATA-GROUP>
    <DATA ref="#dynamic.clickstream"/>
    <DATA ref="#dynamic.http"/>
    <DATA ref="#dynamic.searchtext"/>
    <DATA ref="#dynamic.cookies"/>
   </DATA-GROUP>
  </STATEMENT>
  <STATEMENT>
   <NON-IDENTIFIABLE/>
   <PURPOSE><pseudo-analysis/></PURPOSE>
   <RECIPIENT><unrelated></RECIPIENT>
   <RETENTION><stated-purpose/></RETENTION>
   <DATA-GROUP>
    <DATA ref="#dynamic.http"/>
    <DATA ref="#dynamic.searchtext"/>
   </DATA-GROUP>
  </STATEMENT>
 </POLICY>
</POLICIES>
```

Listing C.2: The P3P form of the Google Privacy Policy

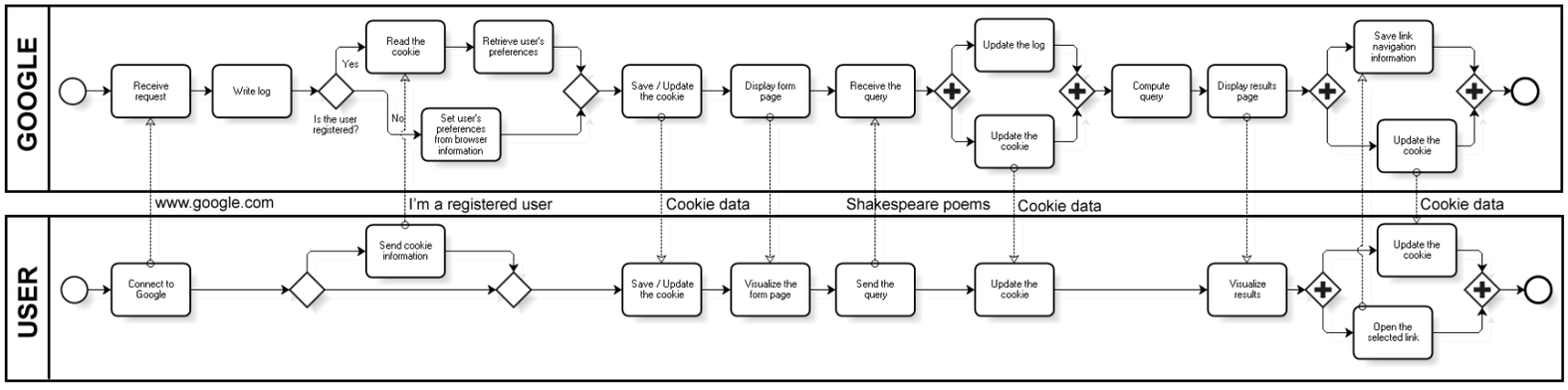


Figure C.1: BPMN representation of a user connecting to a search engine to perform a query

Most images in this thesis have been produced using the BizAgi free tool. Other images were made with our BPeX editor as well as using Microsoft Visio stencils along with other tools.