

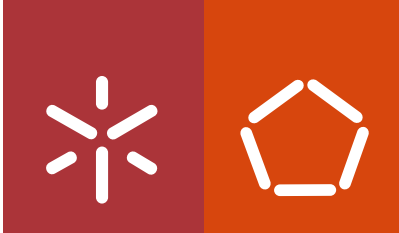


Universidade do Minho
Escola de Engenharia

Maria Estrela Ribeiro Ferreira da Cruz

**Derivation of Data-Driven Software Models
from Business Process Representations**

julho de 2016



Universidade do Minho
Escola de Engenharia

Maria Estrela Ribeiro Ferreira da Cruz

Derivation of Data-Driven Software Models from Business Process Representations

Tese de Doutoramento Tecnologias e Sistemas de Informação

Trabalho efetuado sob a orientação do
Professor Doutor Ricardo J. Machado
e da
Professora Doutora Maribel Y. Santos

julho de 2016

STATEMENT OF INTEGRITY

I hereby declare having conducted my thesis with integrity. I confirm that I have not used plagiarism or any form of falsification of results in the process of the thesis elaboration.

Conduct of the University of Minho.

University of Minho, 21 de Junho 2016

Full name: Maria Estrela Ribeiro Ferreira da Cruz

Signature: Maria Estrela Ribeiro Ferreira da Cruz

Acknowledgements

I would like to thank every person that helped and supported me during this PhD research work. In particular, I would like to express my gratitude to:

- Prof. Ricardo J. Machado and Prof. Maribel Y. Santos for their supervision, guidance and support since the beginning of this PhD journey.
- Prof. João Álvaro Carvalho, from Department of Information Systems, University of Minho, and to Prof. Alexandre Bragança from the Polytechnic Institute of Porto (IPP), for providing supportive comments about this PhD research work and for their ideas.
- Prof. Rui da Silva Gomes, from the Polytechnic Institute of Viana do Castelo (IPVC), where I work since 2007, for the encouragement and the initial thrust.

Finally I would like to thank to all my family for their understanding.

A very special thanks goes to my husband Miguel and to my son André for their support, understanding and encouragement. I want to dedicate this work to them.

Abstract

Derivation of Data-Driven Software Models from Business Process Representations

Organizations are constantly being challenged with new demands imposed by markets and have to respond to new requirements imposed by governments. Organizations need to be prepared to face those challenges and demands in order to survive. Business Process Management (BPM) allows organizations to know themselves and to be prepared to fight new challenges and easily adapt to new situations. BPM is being seen as a key for innovation helping in the simulation of possible scenarios. For these and other reasons, business process management and modeling is being increasingly used by organizations.

A business process model usually is created under the supervision, clarification, approval and validation of the business stakeholders. Thus, a business process model is a proper representation of the reality, having lots of useful information that can be used in the development of the software system that will support the business.

Several authors proposed approaches to derive software models based on business process models. Nevertheless, the generation of a data model based on business process models has been constantly ignored mostly because business process models did not support, until recently, the identification of the persistent data. However, interest in the data and its preservation have increased in the BPMN (Business Process Model and Notation) most recent version, which allows identifying the persistent data manipulated within business processes.

This research work presents and discusses two approaches to derive a data model from business process models: directly, by piecing together information from a set of business process models; and indirectly, by adapting the 4SRS (Four Step Rule Set) method to generate a logical software architecture from business process models and extending it to derive the data model from the logical software architecture. The direct approach is suitable to deal with complete business process models, whereas the indirect approach is more suitable to deal with complex systems, being prepared to detect incomplete business process models and to complete the information derived from business process models with information from other sources.

The derived data model will serve as a basis for the software development, helping reducing time and efforts spent in software design, ensuring the alignment between the software and the business processes, and enabling traceability between the elements in software models and the corresponding business process models.

Resumo

Derivação de modelos de software orientados a dados a partir de representações de processos de negócio

O aumento da dimensão, complexidade e exigências impostas às organizações tem levado a que estas optem, cada vez mais, pela gestão dos processos de negócio (BPM - *Business Process Management*). A gestão dos processos de negócio permite a otimização e agilização dos processos de negócio dando possibilidade às organizações de se adaptar com mais facilidade e rapidez às alterações impostas pelas exigências de mercado. A gestão dos processos de negócio é considerada crucial para a inovação uma vez que permite simular possíveis cenários.

A modelação dos processos de negócio é, normalmente, feita sob a supervisão, clarificação, validação e aprovação dos *stakeholders* de negócio. Desta forma, podemos afirmar que os modelos dos processos de negócio são uma correta representação da realidade, tendo muita informação útil que pode ser usada no desenvolvimento do software de suporte ao negócio.

Vários autores propuseram abordagens para derivar modelos de software baseados em modelos de processos de negócios. No entanto, a geração do modelo de dados com base nos modelos de processos de negócio têm sido negligenciada principalmente porque os modelos de processos de negócios não suportavam a identificação dos dados que devem ser armazenados de uma forma persistente. No entanto, nota-se um crescente interesse nos dados e na preservação destes, na versão mais recente da linguagem BPMN (*Business Process Model and Notation*), a qual permite a identificação dos dados persistentes que são manipulados nos processos de negócio.

Este trabalho de investigação apresenta e discute duas abordagens para gerar o modelo de dados com base na informação existente nos modelos de processos de negócio: uma abordagem direta, que agrega toda a informação existente num conjunto de processos de negócio; e uma abordagem indireta, que adapta o método 4SRS (*Four Step Rule Set*) para gerar a arquitetura lógica com base num conjunto de modelos de processos de negócio e estende-o para gerar o modelo de dados com base na arquitetura lógica gerada. A abordagem direta é mais apropriada para lidar com modelos de processos de negócio completos. A abordagem indireta é mais apropriada para lidar com sistemas complexos e com modelos de processos de negócio incompletos, uma vez que permite complementar a informação extraída desses modelos com informações provenientes de outras fontes de informação.

O modelo de dados gerado serve como base para o desenvolvimento de software, assegurando o alinhamento entre o software e os processos de negócio, e possibilitando a rastreabilidade entre os elementos nos modelos de software e os elementos correspondentes nos modelos de processos de negócio.

Contents

Acronyms	xiii
List of Figures	xv
List of Tables	xvii
1 Introduction	1
1.1 Research Motivation	4
1.2 Research Questions	5
1.3 Research Objectives	6
1.4 Research Method	7
1.5 Overview of the Document	13
2 Notations and Languages for Modeling Information Systems	15
2.1 Introduction	15
2.2 Business Process Management	16
2.3 Business Models	19
2.4 Business Process Modeling	20
2.5 Modeling Languages	21
2.6 Final Remarks	46
3 Deriving Models in Information Systems Development	49
3.1 Introduction	49
3.2 From Business Process Models to Software Models	50
3.3 From Business Process Models to Data Models	53
3.4 From Business Process Models to Use Case Models	55
3.5 From Use Case Models to other Software Models	57
3.6 Use Case Models Decomposition	60
3.7 The 4SRS method	62
3.8 Final Remarks	64
4 Deriving a Data Model from Business Process Models	67
4.1 Introduction	67
4.2 Deriving a Data Model from one Business Process Model	70
4.3 Demonstration Cases with one Business Process Model	73

4.4	Deriving a Data Model from a Set of Interrelated Business Process Models	78
4.5	Demonstration Case with a Set of Business Process Models	83
4.6	Analyzing the Results	88
4.7	Final Remarks	90
5	Deriving a Data Model from a Logical Software Architecture	93
5.1	Introduction	93
5.2	From Business Process Models to a Use Case Model	95
5.3	Getting Use Case Descriptions of the Nobel Prize Demonstration Case	109
5.4	Adapting and Extending the 4SRS to Derive a Data Model	111
5.5	Demonstration Case Aggregating a Set of Business Process Models	119
5.6	Final Remarks	122
6	Deriving a Logical Software Architecture from Business Process Models	125
6.1	Introduction	125
6.2	Extending the UML 2.5 Use Case Meta-model	127
6.3	The Decomposition Triangle approach	130
6.4	The 4SRS Method and the Decomposition Triangle	141
6.5	Deriving a Use Case Model from a Set of Interrelated Business Process Models	143
6.6	Demonstration Case Aggregating a Set of Business Process Models	147
6.7	Final Remarks	150
7	Conclusion and Future Work	153
7.1	Overview of the Undertaken Work	153
7.2	Results and Contributions to the State of the Art	157
7.3	Future Work	159
	References	161
	Appendix	179
A	Use Cases Model of the Library Demonstration Case	179

Acronyms

AD Activity Diagram (UML)

ARIS ARchitecture of Integrated Information Systems

BDV Business Domain View

BI Business Intelligence

BPD Business Process Diagram

BPEL Business Process Execution Language

BPEL4WS Business Process Execution Language for Web Services

BPM Business Process Management

BPMDS Business Process Management Systems

BPMI Business Process Management Initiative

BPMN Business Process Model and Notation

BRV Business Requirements View

BTV Business Transaction View

B2B Business-to-Business

CORBA Common Object Request Broker Architecture

CREWS Co-operative Requirements Engineering With Scenarios

DEMO Dynamic Essential Modelling of Organization

DFD Data Flow Diagram

DSR Design Science Research

EPC Event-driven Process Chains

ERD Entity-Relationship Diagram

IDEF Integrated Definition Methods

ISO/IEC ISO - the International Organization for Standardization and the International Electrotechnical Commission

IT Information Technology

MDD Model Driven Development

MOF Meta Object Facility

NL Natural Language

OMG Object Management Group

OCL Object Constraint Language

OO Object-Oriented

QVT Query View Transformer

SPEM Software & Systems Process Engineering Meta-Model Specification

UML Unified Modeling Language

UMLEWM UML Extended Workflow Metamodel

UMM UN/CEFACT's Modeling Methodology

UN/CEFACT United Nation's Centre for Trade Facilitation and Electronic Business

UP Unified Process

WFM Workflow Management

WfMC Workflow Management Coalition

WSFL Web Services Flow Language

XLANG Web Services for Business Process Design

XML Extensible Markup Language

XPDL XML Process Definition Language

4SRS Four Step Rule Set

List of Figures

1.1	The Build-Evaluate cycle (adapted from (Hevner et al., 2004)) . . .	10
1.2	DSR activities (adapted from (Peppers et al., 2006))	11
1.3	Research phases	12
2.1	BPM life cycle ((van der Aalst, 2004; Ko, 2009))	18
2.2	The business process modeling process (adapted from (Ko, 2009))	20
2.3	A Class diagram example	25
2.4	A UML use case diagram example	28
2.5	Activity Diagram example (adapted from (Silva and Videira, 2005))	29
2.6	Process diagram example (adapted from (Eriksson and Penker, 2000))	31
2.7	A BPMN choreography example (extracted from (OMG, 2011a)) .	33
2.8	Data representation elements (adapted from (OMG, 2011a)) . . .	35
2.9	A Petri-net example (adapted from (Dong and Chen, 2005)) . . .	37
2.10	Example of a Little-JIL diagram (extracted from (Wise et al., 2000))	39
2.11	A XPDL file excerpt (extracted from (van der Aalst, 2003)) . . .	43
2.12	A BPEL file excerpt (extracted from (Juric, 2015))	44
4.1	A BPMN process example (adapted from (OMG, 2011a))	73
4.2	Business process diagram focused on data	74
4.3	Doctor’s Office Data Model	75
4.4	The Nobel Prize Process Diagram (adapted from (OMG, 2010a))	76
4.5	Nobel Prize Data Model	77
4.6	Receiving information from a participant	81
4.7	Sending information to a participant	81
4.8	Relating two data stores	82
4.9	Exclusive decision gateway example	83
4.10	Exclusive merging gateway example	83
4.11	Register User business process model	84
4.12	Lend a Book business process model	84
4.13	Reserve a Book business process model	85
4.14	Renew a Loan business process model	85
4.15	Return a Book business process model	85
4.16	Penalty treatment business process model	87

4.17	The resulting data model	88
5.1	The Nobel Prize Use Case Diagram	98
5.2	splitting and merging gateways	102
5.3	The 4SRS table (an excerpt)	116
5.4	The resulting logical architecture	117
5.5	The resulting data model	120
5.6	The resulting logical architecture	121
5.7	The resulting library data model	122
6.1	Extended UML use case meta-model	129
6.2	The first iteration result	132
6.3	The second iteration result	134
6.4	The library system decomposition triangle	137
6.5	“{1.1} Select books to acquire” use case diagram (<i>level 3</i>)	138
6.6	A tree structure example	142
6.7	The 4SRS transformation	142
6.8	A generic decomposition scheme	145
6.9	Actors diagram (level 0)	147
6.10	Use Case diagram (level 1)	147
6.11	Purchase a Book business process model	148
6.12	Lend a Book business process model	148
6.13	Return a Book business process model	148
6.14	Penalty Treatment business process model	148
6.15	Complete Use Case model (level 3)	149
A.1	A use case diagram of the Library Demonstration Case	183

List of Tables

2.1	Different perspectives of business processes modeling languages (based on (Giaglis, 2001))	47
3.1	Summary of approaches that make the relationship between models	65
3.2	Summary of approaches that obtain the UML use cases model . .	66
4.1	The Data handling	69
4.2	Entities and Relationships	87
4.3	Entities manipulation	89
5.1	The template for describing use cases	99
5.2	The use case sentences originated by Data Associations	101
5.3	The use case pre-condition originated by gateways	103
5.4	Generic sentences originated by events	104
5.5	The sentences originated by Start and Catching events	105
5.6	The use case descriptions originated by End and Throwing events	106
5.7	The use case descriptions originated by Intermediate Interrupting events	107
5.8	The use case descriptions originated by Intermediate Non-Interrupting events	108
5.9	Send Nomination Form use case description	109
5.10	Send List of Preliminary Candidates use case description	110
5.11	Submit Report Recommendations use case template	110
5.12	The descriptions of the use cases using the defined template (part I)	112
5.13	The descriptions of the use cases using the defined template (part II)	113
6.1	Use cases descriptions (<i>level 1</i>) - first proposal	133
6.2	Use cases description (<i>level 1</i>) - second iteration	135
6.3	“{1} Purchase books” use cases description (<i>level 2</i>)	136
6.4	“{1.1} Select books to acquire” use cases description (<i>level 3</i>) . .	139
7.1	Comparing the two presented approaches to derive a data model from business process models	156

Chapter 1

Introduction

Markets globalization and the constant increase of competition between companies demand constant changes in organizations in order to adapt themselves to new circumstances and to implement new strategies. Organizations need to have a clear notion of their internal processes in order to increase their efficiency and the quality of their products or services, enhancing the benefits for their stakeholders¹ (Schmiedel and vom Brocke, 2015; van der Aalst, 2015).

According to Schmiedel and Brocke, business process management can be seen as a key for innovation (Schmiedel and vom Brocke, 2015) helping companies and organizations to simulate possible scenarios. For this and other reasons, many organizations adopt a business process management (BPM) approach (Batoulis et al., 2015). BPM includes methods, techniques, and tools to support the design, enactment, management, and analysis of the operational business processes of an organization (van der Aalst, 2004).

A business process is a set of interrelated activities that are executed by one organization to create a product or service (Hammer and Champy, 2001). It is important to note that activities belonging to a business process are not performed by a single individual or department, but typically involve many people, machines and systems of an organization or different organizations, working together to achieve a common business purpose (Ko, 2009).

The BPM supports the business processes using methods, techniques and software to design, approve, monitor and analyze operational processes involving people, organizations, applications, documents and other information sources (ter Hofstede et al., 2003). The BPM can be considered as an extension to the Workflow Management (WFM) classic method, which is, according to the Workflow Management Coalition (WfMC), an automation of a business process in which information passes from one participant to another, suffering actions according to a set of procedural rules (Coalition, 2011; Ko, 2009; Weske et al., 2004; Alter, 2015).

¹Who affects or can be affected by an organization's actions.

There are several languages and tools that can be used to model business processes such as Petri nets (Weske, 2012), EPC (Event-driven Process Chains) (van der Aalst, 1999; Kindler, 2004), IDEF (Integrated Definition Methods) (Aguilar-Savén, 2004), BPMN (Business Process Model and Notation) (OMG, 2011a; Allweyer, 2010), some extensions of UML (Unified Modeling Language), such as the Eriksson and Penker extension (Eriksson and Penker, 2000), among others. In this research the BPMN language, currently in version 2.0 (OMG, 2011a), is selected because it is a widespread OMG (Object Management Group) standard that is actually used both in academia and in organizations, and is a language easy to understand and usable by people with different roles and training from top managers to Information Technology (IT) professionals (OMG, 2011a; Magnani and Montesi, 2009; Braun and Esswein, 2014). According to Andreas Meyer, BPMN is a modeling language well accepted in companies and that receives the influence from them, as is the case of SAP, Unisys, Oracle and Software AG (Meyer et al., 2011).

The constant changes imposed to business processes demands adaptations to the software that supports the business. For many years, the software development teams faced serious difficulties in compiling the list of requirements. Like Pankaj Jalote states, “the software often does not do what is supposed to do or does something it is not supposed to do” (Jalote, 2008). This is because, on the one hand, companies create systems to support their business, on the other hand, software engineers, focused on the development process, usually define the functional requirements list based on the users’ needs. Other indicated reasons for software projects failure are the poor understanding of the business by the software engineer, and the fact that business owners do not understand the language used by software engineers debilitating the understanding between them (Jalote, 2008; Barjis, 2008).

If on one side the business process management and modeling are increasing their relevance, on the other side the software development teams still have serious difficulties in performing elicitation and defining the applications requirements (Jalote, 2008; Barjis, 2008; Redlich et al., 2014; van der Aa et al., 2015). In fact, one of the main software quality objectives is to assure that a software product meets the business needs (Jalote, 2008). For that, the software product requirements need to be aligned with the business needs, both in terms of business processes and in terms of the informational entities those processes deal with. This drives us to the question: “How can Business Process Models be used as a basis to design the software applications’ data model that supports the business?”.

Researchers and professionals in information systems have recognized that understanding the business process is the key for identifying the user requirements of the software that supports it (Mili et al., 2003; Scott, 2007; Cardoso et al., 2009). True cost-benefits advantages can only occur when software processes are aligned with organizational processes (Russell et al., 2006). As Van der Aalst wrote in (van der Aalst, 2004), about business process support, “to

support business processes an enterprise information system needs to be aware of these processes and their organizational context”. However, the tasks of business process analysis and software development are managed by different groups of people and typically use different languages.

Giaglis says that although the benefits of aligning business process with their information systems modeling is studied in theory, such integrated design strategies have rarely been implemented (Giaglis, 2001; Cardoso et al., 2009). Traditionally, business analysts and information systems professionals perform distinct roles within organizations, each equipped with its own tools, techniques, skills and even different terminology (Giaglis, 2001; van der Aa et al., 2015).

Ko states that, for people who design and maintain information systems that support BPM, within and between enterprises, it is very beneficial and advantageous to understand the fundamentals of the BPM discipline (Ko, 2009; Cardoso et al., 2009).

One of the advantages of business process modeling is that it is understood by all actors involved, and this allows to achieve results closer to reality (Dorn et al., 2009). According to Yue *et al.*, business experts and software experts need to use a common language, to communicate, and to understand each other. That language can be the BPMN language. The BPMN is a complete language, easy to learn and easy to understand (Yue et al., 2011).

Once the BPM already analyzes the business processes, conducting meetings with stakeholders, using a language and a notation understood by everyone, why not using the information obtained during this process (modeled in business process models) as the basis for the identification of the data needs of the software that supports these businesses? This PhD research aims to create a systematic approach able to generate a data model based on the set of business processes (modeled in BPMN) that will be supported by the software under development.

Ideally, in a Model Driven Development (MDD) context, some software model aspects should be derived from the existing aspects of business process models of an organization (Paradkar and Sinha, 2015).

When we are dealing with a business process it is inevitable to mention the data involved, or the information that flows throughout the process. So, to enable process control and business supporting software development, that information must be stored. However, as referred by OMG, data modeling is not a BPMN 2.0 goal (OMG, 2011a). Nevertheless, data is a key component whose relevance has increased, not only as a support to the business itself, but also for Business Intelligence (BI) operations (Meyer et al., 2011). Therefore, the data model is a fundamental model for designing software applications. Van der Aalst states that the data collected during a process execution can be used to “analyze running processes, discover bottlenecks, waste, and deviations” (van der Aalst, 2015).

The main research motivations are presented in the next section.

1.1 Research Motivation

The motivation for this PhD research work is based on the following observations, which can be drawn from the study summarized in chapter 3:

- The number of organizations that use the business process management and modeling is increasing (Aguilar-Savén, 2004; Ko, 2009; Kalenkova et al., 2014; Alter, 2015; Schmiedel and vom Brocke, 2015).
- Information systems researchers and professionals have recognized that understanding a business process is the key to identify the user needs of the software that supports it (Mili et al., 2003; Shishkov et al., 2002; Scott, 2007).
- It is recognized that the business process modeling shall provide the basis for the development of the software products that support the business (Yue et al., 2009). Ideally, and in the context of the model-based software development, some software model aspects should be able to be derived from existing aspects of the business process models (Paradkar and Sinha, 2015).
- Some of software development processes, like Unified Process, already refer to business process modeling as a pre-requisition to the next steps of software development (Štolfa and Vondrák, 2008; Jacobson et al., 1999).
- The languages used by business process modelers are not the same languages used by software modelers (Cardoso et al., 2009).
- Software development still reveals difficulties, and spends too much time, in identifying user requirements and in the creation of software design models, which are based in different sources of information and sometimes prove to be inconsistent and even contradictory between them (Dobing and Parsons, 2006; Jalote, 2008; van der Aa et al., 2015).
- Within the various modeling languages, BPMN is an OMG standard very widespread and effectively used in business process modeling in companies (Meyer et al., 2011; Muehlen and Recker, 2008; Kalenkova et al., 2014; Braun and Esswein, 2014; Kocbek et al., 2015).
- BPMN is consolidating the position of the language used by default in the business process modelling field (Kocbek et al., 2015; Baumann et al., 2015).
- When it comes to activities involved in the business process it is inevitable to talk about the data involved, or about the information that circulates

throughout the processes. Therefore, it is necessary to store the vast majority of information involved both, for process analysis and for the development of applications that support the business. Thus, approaches covering the data modeling will be very useful since the data model is one of the most important software development models and none of the current approaches generates a complete data model based on the existing information in business process models (Brdjanin and Maric, 2013).

- There are several approaches that link business process modeling and software modeling. However, none of the approaches generates the data model based on business process models. Moreover and most importantly, to our knowledge, none of the approaches aggregates in a data model the data information that can be extracted from the set of business process models that will be supported by the software under development.

In short, from a software development point of view, it makes sense to think of an approach that, based on a set of interrelated business processes models, extracts useful information in order to define the data model used in the development of business supporting software.

All the approaches cited in chapter 3, generating software models from business process models, base their analysis in only one business process model. But, typically, in a real situation, a software product does not support only one process, but a reasonable set of processes. So, in order to generate useful software models, it will be necessary to consider the set of business process models that will be supported by the data model of the software product in development.

1.2 Research Questions

A business processes model, in BPMN, allows identifying information like: the activities performed in the process; who performs each activity; in what circumstances the activities are performed; what resources are involved; who is involved in the processes; which data circulates through the process and which is the provenance of the data; which data should be stored in a persistent manner, among other content. Thus, will it be possible to model these data in a useful perspective for developing software that supports the business?

In fact, the business processes modeling is increasingly used and disclosed. At the same time, the software development continues to have problems in designing software supporting systems and continues to spend much time and effort to elicit relevant requirements. At this point the following questions can be posed:

1. How to obtain the data model of the software products that support the business from a set of business process models?

Despite the data structure being considered, in most cases as the most important part of a software product, there are few authors that present approaches to get the data model from the business process models. And the existing ones, base their approach in only one business process model.

The authors who focus on this issue, as (Sturm, 2008; Brambilla et al., 2008; Magnani and Montesi, 2009), faced problems, especially with the inability to identify the data that must be stored in a persistent manner.

In BPMN, this obstacle was overcome with the inclusion of the new graphical element *data store* on the latest release (BPMN 2.0). The *data store* allows identifying the data that must be maintained in a persistent manner, i.e., data that must remain beyond the process life cycle (Allweyer, 2010; OMG, 2011a).

2. Is it possible to directly obtain the data model from a set of business process models?

The BPMN most recent version allows identifying the data that must be kept in a persistent manner. However, the approach must be able to identify not only the entities but also the entity attributes and the relationship between those entities. Information about relationships is not modeled in business process models, so rules must be defined to derive those relationships.

3. Is it possible to create an approach that allows complementing the information obtained from business process models with information from other sources and generate a proper data model in high complexity information systems?

Some studies reveal that, usually, business process models have bad quality or are incomplete (Weber et al., 2011). Hence, the existing information in the BPMN models may not be enough to directly generate a complete data model. The 4SRS (Four Step Rule Set) is an iterative method that incrementally verifies and validates the elicited requirements modeled as use cases, and creates a logical software architectural model (Machado et al., 2006). The 4SRS has proved to be capable of dealing with complexity (Machado et al., 2006; Ferreira et al., 2012) and allows detecting and completing lacking information.

1.3 Research Objectives

In the context of model-based software development, some aspects of the software model should be derived from existing features in the model of the business processes of an organization.

This PhD research work intends to achieve two main objectives:

1. Definition of an approach, including a set of heuristics and rules, for generating a data model based on a set of interrelated business process models that will be supported by the software under development.

Often data is modeled as a support to the requirements previously defined. However, the data is, in most cases, a main element of a software product. This work intends to create a systematic approach to derive a data model that includes all data that must be maintained in a persistent manner, and that provides support to the requirements previously defined and identified based on the same set of business process models.

2. Definition of a method, comprising a set of heuristics and rules, that is able to obtain a data model from the logical software architecture derived from the information existing in a set of interrelated business process models.

The 4SRS is a method that generates the logical software architecture from use case models. This work intends to adapt and extend the 4SRS in order to work with the information existing in a set of business process models and to derive the data model. To do that, it is intended to create an approach to generate the use case model, including descriptions, based on the set of business process models that are intended to be supported by the software application in development, thus ensuring the implementation of all requirements that come directly from the business process models. The generated use case model is used to feed the 4SRS tabular transformation. After that, another approach will be created to derive a data model from the logical software architecture.

The generated models must be consistent with each other, meaning that the data entities referred to in the use case model must be represented in the data model. This way, business and software modeling efforts can be joined together, reducing the analysis time and avoiding forgetting functional or even data requirements. To be possible, business process modelers should include in business process models all relevant information. More precisely, to obtain a complete data model, the business process model should contain the information about all data involved in the processes including the identification of the persistent data.

1.4 Research Method

There is a large number of research methods. Nevertheless, as indicated by Avison *et al.*, the best methods depend on the research topic and on the research questions addressed (Avison *et al.*, 1999).

In the information systems area, research often has a practical nature, which is often the application of theories traditionally used in other areas such as economics, management or computer science, to solve problems that intersect Information Technology (IT) and organizations (Peffer *et al.*, 2006).

In the information systems discipline design is crucial. To March and Storey, information systems professionals are typically engaged in the design and implementation of IT artifacts that improve the business performance of organizations and might have an impact on economic gains (March and Storey, 2008). In this type of research IT artifacts that extend the known boundaries of IT are developed, addressing important issues that hitherto no one has thought to computerize (Hevner et al., 2004; March and Storey, 2008).

This PhD research work is structured according to the Design Science Research (DSR) methodology. The DSR methodology is usually applied in the design and development of artifacts, in order to solve specific problems of organizations, giving possibility to, directly or indirectly, increase their profits (Hevner et al., 2004). Thus, a research project that uses DSR requires the intentional creation of an innovative artifact to solve a specific problem in a given domain (Hevner et al., 2004; Carvalho, 2012).

An IT artifact is created to allow a representation, analysis, understanding (March and Storey, 2008) and development of information systems success within an organization.

Research projects using the DSR in information systems produce basically four types of IT artifacts (March and Storey, 2008; Vaishnavi and Jr., 2008): constructs, models, methods and instantiations.

- **Constructs** - a constructor is a set of vocabulary and symbols used to define the problem and to specify the solution. This set allows the definition of a language that can be used to communicate knowledge within the area (March and Smith, 1995). The language definition has a strong impact on how the tasks and problems are designed. The constructors arise in the problem design and can be refined throughout the design cycle.
- **Models** - a model is a set of propositions that express the relationship between constructors (March and Smith, 1995). Models use constructs to represent situations (Hevner and March, 2003). A model is a representation of the real-world. As stated by Giaglis, modeling allows filtering the real-world complexity aimed to direct the effort to the important and more relevant aspects (Giaglis, 2001). A model should be able to capture the structure of reality so that their representation becomes useful (Peffer et al., 2006). Thus, models help to understand the problem and to relate the problem to the solution.
- **Methods** - a method is a set of steps used to perform a task (March and Smith, 1995). A method is a procedure that provides a guide to the problem resolution (Hevner et al., 2004). A method is based on the constructs and models previously defined. A method can be linked to a specific model where the steps are entry points into the model, or can be used to transform (or translate) into another model. The methods can be formal, specified by

using mathematical methods or algorithms, or informal, specified through approaches that are textually or graphically described. The obligation to use certain methods can influence the definition of the constructs and models developed.

- **Instantiations** - an instantiation is the operationalization of the constructs, models and methods of a system (Vaishnavi and Jr., 2008). An instantiation is the realization of the artifact in its environment. Instantiation involves a full articulation of language and models and embodying methods (March and Smith, 1995). Therefore, an instantiation can demonstrate the feasibility and effectiveness of the models and methods involved. The instantiations provide operational artifacts that can be used to study and improve new ideas and developments in the area.

The artifacts are often built in the form of prototypes. The artifact shall be innovative and useful for problem solving, so its evaluation is essential (Hevner et al., 2004).

To Hevner *et al.*, DSR forwards the research by building and evaluating the designed artifacts (Hevner et al., 2004). The artifacts are built to meet the business needs. In (Venable, 2006; Peffers et al., 2006; March and Smith, 1995), the authors opinions are that research in design science is essentially a problem-solving paradigm. In this type of research, there are two activities that deserve special attention, namely to build and evaluate:

- **Develop/Build** - refers to the artifact design and construction. The artifact is built to perform a specific task. The construction shows that the artifact is feasible. Once built, the artifact can become an object of study. Constructs, models, methods and instantiations are a technology that, once built, can and should be scientifically evaluated (March and Smith, 1995).
- **Justify/Evaluate** - refers to the definition of criteria and evaluation metrics. Also refers to assessment of the artifact performance with respect to defined criteria. Metrics define what one intend to perform or improve. A metrics failure may lead to a failure in the performance measurement artifact and thus a misjudgment (March and Smith, 1995). In addition to the criteria and evaluation metrics it is necessary to ensure an appropriate environment for the evaluation. The environment can include all technical infrastructures and data needed to the assessment. Finally, it is necessary to identify the situations and environments in which the artifact works, in what situations it does not work, and it is necessary to identify how and why this happens. This leads to a deeper and detailed knowledge about the artifact which can lead to the creation of new ideas to change the artifact. An IT artifact can be evaluated in terms of functionality, completeness, consistency, accuracy, performance, reliability, usability, fit for the organi-

zation’s requirements, and other relevant quality attributes (Hevner et al., 2004). For J. Carvalho, the success of an artifact can be measured by its usability, efficiency and effectiveness (Carvalho, 2012). When the artifact refers to a software product, ultimately, its success can be assessed by using the ISO/IEC² 25010 quality attributes: functional suitability, performance and efficiency, compatibility, usability, effectiveness, reliability, security, maintainability, etc. (IEC, 2010).

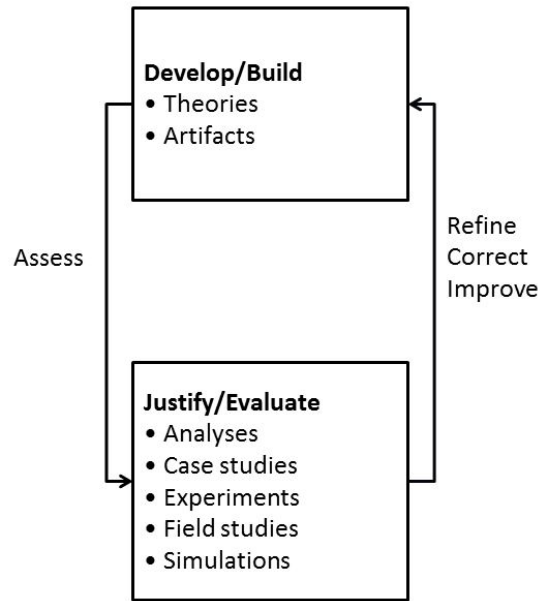


Figure 1.1: The Build-Evaluate cycle (adapted from (Hevner et al., 2004))

As shown in Figure 1.1, the Build-Evaluate activities constitute an iterative process. On the one hand, to Justify/Evaluate the artifact can motivate, or suggest changes to its design and construction (Develop/Build). On the other hand, when the artifact design and construction is modified, the artifact should be reevaluated to determine if progress exists. Progress exists when one technology is replaced by a more efficient one (March and Smith, 1995).

The Build-Evaluate cycle of an artifact commonly occurs several times during the research. During the various iterations weaknesses in the artifact and refinement needs can be identified and reevaluated. The development of several iterations forward research to meet the business needs (Hevner et al., 2004).

The artifact construction is complete when it satisfies the problem requirements and constraints that it seeks to solve. However, sometimes redefinition and reevaluation can be proposed for future work (Hevner et al., 2004).

²ISO - the International Organization for Standardization and IEC - the International Electrotechnical Commission

The DSR uses an iterative approach that consists in six main activities (Peffer et al., 2006): problem identification and motivation; definition of objectives for the solution; design and development; demonstration; evaluation and reporting.

Figure 1.2 represents the activities performed during the DSR research process.

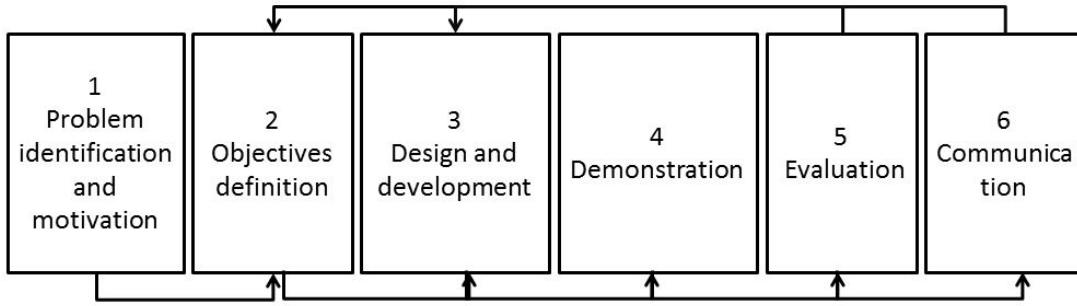


Figure 1.2: DSR activities (adapted from (Peffer et al., 2006))

Process activities can be performed in sequence, i.e. by order, from 1 to 6, or may follow a different order in which some of the activities can be performed several times. The activity 5 (evaluation) or 6 (communication) can induce the research to get back to activities 2 (defining objectives) or 3 (design and implementation).

The created artifacts’ utility, quality and effectiveness must be rigorously demonstrated and assessed. The assessment of an IT artifact requires the definition of metrics and the creation of appropriate environment and data. DSR, through the artifacts’ build-evaluate iterations, guides the research to meet business needs.

This PhD research work aims to create an approach to obtain a data model based on the information existing in a set of business process models serving for the development of the software that will support the business.

To accomplish the research objectives, the steps shown in Figure 1.2 are followed. The first step, the problem identification and motivation, has already been presented herein (section 1.1). The remaining steps have been executed several times. Thus, the investigation is divided in several phases, where each phase is constituted by the steps 2 to 6 shown in Figure 1.2. To each phase specific goals are defined, drawing and implementing, demonstrating, evaluating and communicating the obtained results. The steps that integrate the research are shown in Figure 1.3. In this figure, the solid arrows represent new steps created during this PhD research work. The dashed arrows represent existing steps that were adapted in this research work.

Thus, the research work is divided into the following phases:

1. Explore the possibility to get the data model from one business process

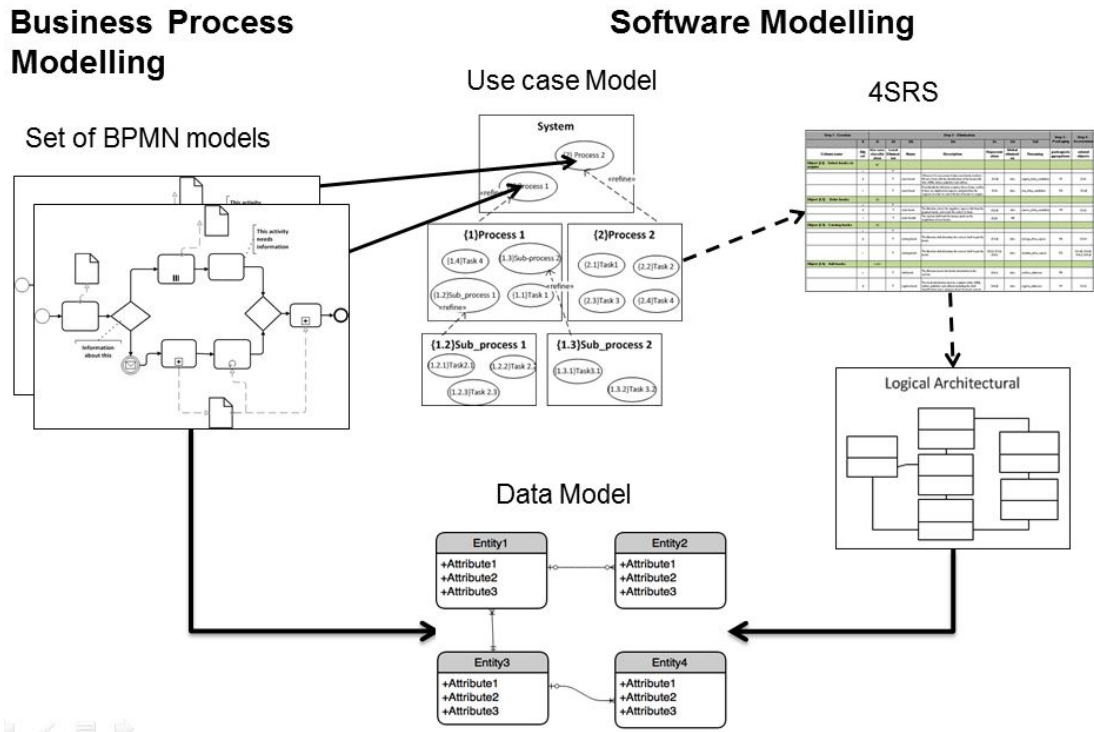


Figure 1.3: Research phases

model, modeled in BPMN language. This approach is presented in chapter 4, section 4.2, and demonstrated in section 4.3.

2. Improve, complete and extend the approach presented in the previous phase in order to aggregate in one data model the information available in a set of interrelated business process models, modeling the business processes that will be supported by the software under development. This approach is presented in section 4.4 and demonstrated in section 4.5.
3. Extend the 4SRS to derive a data model from the logical software architecture. The 4SRS will be adapted to work based on the use case model previously derived from the set of business process models that will be supported by the software under development. This way the data model will be created based on the set of business process models. This approach is presented in section 5.4.
4. Derive the use case model by aggregating all information available in a set of business process models. This use case model will be used to feed 4SRS method. This phase will be divided in three sub-phases:
 - (a) Create an approach to derive a use case model, focused on use case descriptions, from one business process model. A structured and con-

trolled language is defined to be used in use case descriptions. The use case descriptions are “extracted” from the information presented in the BPMN model. This approach is presented in section 5.2.

- (b) Create an approach to refine the UML use case model in order to decompose the use cases to be used by the 4SRS method. The approach, named as “the decomposition triangle”, is presented in section 6.3.
- (c) Define an approach (presented in section 6.5) to integrate in one UML use case model all the information existing in a set of business process models. To derive the use case model “the decomposition triangle” approach is used. This new approach gives special attention to the generation of the use cases descriptions (textual structured information) in order to provide complete information for the 4SRS. This way it will be possible to generate the software logical architecture based on a set of business process models.

Each phase is assessed by applying the created approach in demonstration cases and ends with the communication of the results. Thus, six papers were presented in scientific conferences after being approved through a peer reviewing process.

1.5 Overview of the Document

This PhD dissertation is organized into a total of seven chapters:

Chapter 1 - This is the current chapter. It presents some introductory notes, the motivation and goals of this research work, along with the research questions and objectives and the methodological approach used in this PhD dissertation.

Chapter 2 - This chapter addresses concepts and definitions that are needed throughout the research. It briefly points out software modeling and business process modeling and management concepts. This chapter also presents the software models and the business process models and notations more significant for this research.

Chapter 3 - This chapter presents several approaches for establishing relationships between different software models and business process models. Approaches that approximate the software development area with the business process modeling area are highlighted.

Chapter 4 - This chapter presents two approaches, integrating the research work being presented here, to derive a data model from business process models. The first approach deals with only one business process model, while the second approach aggregates in one data model all the information about persistent data that can be extracted from a set of business process models.

Chapter 5 - This chapter presents an approach to support the construction of use case models based on business process models highlighting the derivation

of the use case descriptions, which are created using a set of predefined structured sentences (in natural language) mapped from BPMN model elements. The generated use case model is used to feed the 4SRS tabular transformation.

In the second part of this chapter the 4SRS is adapted to deal with the derived use case model from business process models and extended to generate a data model supporting the generated logical architecture and the elicited requirements.

Chapter 6 - This chapter presents two more approaches, integrating the research work being presented here: The decomposition triangle approach and an approach to generate a use case model from a set of business process models. The decomposition triangle decomposes and refines UML use case models as a mean to organize and manage use cases to be used by the 4SRS method. The decomposition triangle is then used in the second approach presented in this chapter to aggregate in one use case model all the information that can be extracted from a set of business process models. The approaches presented in this chapter are preparing the requirements (modeled as a use case model) to be used by the 4SRS to generate the logical software architecture.

Chapter 7 - This chapter summarizes the obtained results, presents conclusions and points out future work.

A list of references and an appendix are also included at the end of the document.

Chapter 2

Notations and Languages for Modeling Information Systems

This chapter presents some concepts and definitions that are used throughout the dissertation. It points out business models, business process management, business process modeling, software modeling and the main reasons that lead to the increasing demand of these methodologies by organizations. It also presents several languages that can be used in business processes modeling and in software modeling.

2.1 Introduction

BPM is increasingly being used by organizations as a means to improve their products or services quality and to improve their productivity (van der Aa et al., 2015; Redlich et al., 2014). Business process modelling is used to detect bottlenecks, waste, and deviations and to simulate possible improvements to business processes (Schmiedel and vom Brocke, 2015).

The increasing interest in BPM led to an increasing number of languages and notations used in this area. Some of them, considered as most important, are presented in this chapter.

Fettke *et al.* say that modeling is the main vehicle to analyze, develop and implement information systems (Fettke et al., 2005). However, the modeling process is most of the times, time consuming, resources consuming and an imperfect process. A model is a simplified view of the complex reality. Modeling allows the decision makers to filter the complexity of the real world so that the effort can be directed only to aspects considered important (Giaglis, 2001).

In the software development process several models are used to represent different points of view. Nowadays, talking about software modeling, is talking about UML (Olivé, 2010; Marshall, 1999; Lange et al., 2006). However, when it comes to modeling databases, in most cases, one still speaks of relational databases, represented by the traditional Entity-Relationship Diagrams (ERD)

(Silva and Videira, 2005; Silingas and Butleris, 2008). According to Dobing and Parsons, UML use in information systems development has increased significantly and is now the second most widely used technique, after Entity-relationship diagrams (Dobing and Parsons, 2009).

The remainder of this chapter is structured as follows. Next section presents business process management and its current trends. Section 2.3 presents some examples of business models. Business process modeling is presented in section 2.4. Modeling notations and languages are presented in section 2.5. This chapter ends with a brief discussion about business process modeling languages.

2.2 Business Process Management

BPM helps organizations to improve the perception of their own business, helping to improve the quality of their products or the efficiency of their services and thus increasing the benefits for their stakeholders. This justifies the current trends of BPM.

The business process management, in recent years, has revealed four essential tendencies (Dong and Chen, 2005; Redlich et al., 2014; van der Aa et al., 2015):

1. BPM is becoming more important;
2. The organizations that use BPM are increasing in number;
3. Business processes are becoming increasingly complex;
4. Business processes are subject to constant changes.

But what is exactly Business Process Management? There are various definitions of BPM, including some mentioned below. To White and Miers the business process management is a way of thinking, a management philosophy focused on a continuous improvement of organization's processes (White and Miers, 2008).

In (ORACLE, 2011), BPM is a management model of continuous improvement, which should always be aligned with strategic business objectives, and for that reason, it spans different departments, areas and business units across an organization.

The BPM supports the business processes using methods, techniques, tools and software applications to design, approve, monitor and analyze operational processes involving humans, organizations, applications, documents and other information sources (ter Hofstede et al., 2003; Ko, 2009).

Summing up, BPM manages the business processes. But, what is a business process? A process can be defined as a set of activities performed in parallel or in sequence, related to each other, with a beginning and an end in order to achieve a common goal.

Ryan Ko defines process as a specification of a sequence of work activities over time and place, with a beginning and end, and inputs and outputs clearly identified (Ko, 2009).

In the context of software development, a process specifies how the activities are organized, managed, measured, supported and improved to achieve a goal (Estublier, 2006). In the context of business, a business process is a set of related activities, developed by one organization for the creation of their products or the execution of their services (Hammer and Champy, 2001).

To Aguilar-Savén a business process is the combination of a set of activities within a company, with a structure that describes their logical order and dependencies, whose goal is to produce a desired result (Aguilar-Savén, 2004).

It is important to note that a business process, being a structured sequence of specific activities, is not performed by a single individual or department, but it involves several people, machines and systems of an organization or different organizations to work together to achieve one common business goal (Ko, 2009).

The business processes can be divided into two major groups (Ko, 2009; OMG, 2011a):

- **Private Business Processes** - A private process is a process internal to a specific organization. It represents processes internal to the company, which include strategic management processes, and processes at the operational level;
- **Public Processes** - A public process represents the interactions between a private Business Process and other Processes or Participants. They are also known as collaborative business processes, involving the participation of external organizations, for example, delivery of goods, materials requirements, etc..

Information technologies (IT) are, often, used to support the management of business processes. Software tools that support the business processes management became known as Business Process Management Systems (BPMS). The BPMS software systems are generic, guided by explicit process plans to manage operational business processes (ter Hofstede et al., 2003). The system should be generic and process-oriented to allow constant changes to the processes it supports. Thus, the BPM is a cycle that is constantly changing.

In general the authors are unanimous in what concerns to BPM lifecycle main steps. As shown in Figure 2.1, the BPM life cycle involves four essential steps (Ko, 2009; Weske, 2012; Aguilar-Savén, 2004; van der Aalst, 2004):

1. **Process Modeling** - at this stage the business processes are modeled using a Business process modeling language like BPMN or other; Usually the BPM requires the modeling of an existing (“as-is”) or desired (“to-be”) process. The process models may include several perspectives like control-flow, data-flow, etc.;

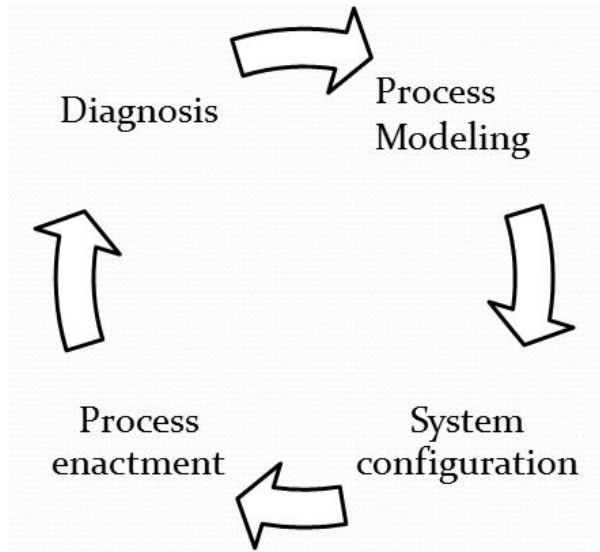


Figure 2.1: BPM life cycle ((van der Aalst, 2004; Ko, 2009))

2. System configuration - at this stage, based on the process design, the process-aware enterprise information system is realized. The BPMS is configured.
3. Process enactment - at this stage, the designed business processes are deployed;
4. Diagnosis - at this stage the business processes (and activities) are analyzed and monitored. The BPM analysis can identify improvements, bottlenecks and potential gaps in business processes. This way, organizations can improve performance, support new processes, explore new technologies, etc. in a controlled way.

After identifying the weaknesses and potential improvements, the processes are redesigned, and the cycle continues.

A wide variety of paradigms and methodologies of the organizational management theory, computer science, mathematics, linguistics, semiotics and philosophy were adopted, making the BPM a transversal discipline (Ko, 2009).

Over the last decades it was noticed an increasing number of standards, new models and new tools to model business processes. BPM tools are used to create and validate the business process models and to store it in databases so that they can be updated whenever necessary (Delgado et al., 2015). BPM tools can be also used to provide meaningful metrics to business analysts for measuring and optimizing business processes (Delgado et al., 2015). These tools allow process managers to change and improve the process whenever necessary and possible (III, 2014).

2.3 Business Models

A business model is an abstract representation of how an organization makes money (representing the organization “as-is”) or how an organization intends to make money (representing the organization “to-be”). According to Osterwalder and Pigneur, a business model describes “how an organization creates, delivers, and captures value” (Osterwalder and Pigneur, 2010).

The business model includes core aspects of a business such as business purpose, target customers, infrastructure, organizational structures, strategies, trading practices, business process, etc. (Osterwalder and Pigneur, 2010). CANVAS and DEMO (Dynamic Essential Modelling of Organization) are two examples of business models.

Jan Dietz defined DEMO as a theory about constructing and operationalizing organizations and is also a methodology for modelling, (re)designing and (re)engineering organizations (Dietz, 2001). CANVAS is one of the most popular framework for business models. The CANVAS business model was applied, tested and successfully used in many organizations to create new business strategic alternatives (Barquet et al., 2011).

Comparing CANVAS business model with DEMO, Steven Alter states that the CANVAS business model is more informal than DEMO. CANVAS produces “brief and informal summaries of business models” and DEMO produces “formal, rigorous, carefully documented, and internally consistent” models (Alter, 2015).

The ARIS (ARchitecture of Integrated Information Systems) is framework for the description of the organizational structure and business processes (Scheer and Nüttgens, 2000). The ARIS framework is composed of the four levels of process engineering, process planning and control, workflow control and application systems (Scheer and Nüttgens, 2000; Bork and Fill, 2014).

The ARIS framework is one of the most used in organizations and is very well accept within business community (Cardoso et al., 2009). Traditionally ARIS and Event-driven Process Chains (EPCs), usually, are used together (Bork and Fill, 2014). Nevertheless, in recent versions of the ARIS Toolset also support the BPMN to model business processes.

According to Dorn *et al.*, the business model research focus has changed over time. It began by creating taxonomies of business models, then passed to the description of the business models elements and then to the construction of business models ontologies (Dorn et al., 2009). These ontologies provide vocabulary and a set of concepts, which is used to describe the business logic that is understood by all stakeholders, facilitating, this way, the understanding between them. The business modeling follows a set of methods and techniques that helps the organization in formalizing their business, offering a uniform representation of the organization. The business model may serve as a business guide to improve how the business is operated (Eriksson and Penker, 2000).

Next section addresses business process modeling.

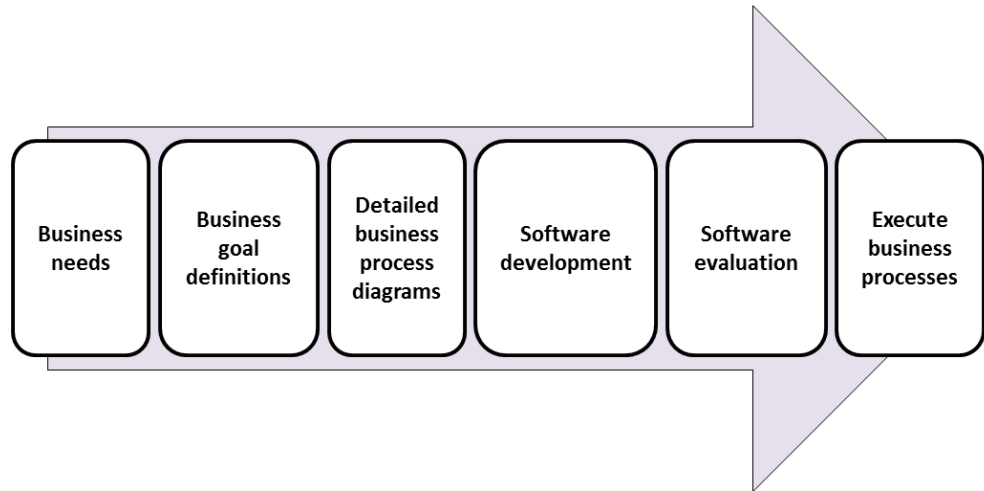


Figure 2.2: The business process modeling process (adapted from (Ko, 2009))

2.4 Business Process Modeling

A model is a simplified view of the complex reality, so modeling allows the decision makers to filter the complexity of the real world so that the effort can be directed only to aspects considered important (Giaglis, 2001). A process model can provide a detailed understanding of a process (Aguilar-Savén, 2004). To Eriksson and Penker, working with models increases understanding about the business and provides a better insight on how to improve business (Eriksson and Penker, 2000). The business process modeling enables a common understanding of the analysis of a business process by all stakeholders. A business process model is an abstract description of the flow of one or more business processes with activities and resources. Resources can be people, equipment, machinery, software, and others (Hammer and Champy, 2001). Contrary to the models used in the software process, which are understood only by specialists, the business process models should serve as the basis for communication between all people involved, so they must be understood by all (Becker et al., 2000).

To Andersson *et al.* a business process model describes and visualizes the transfer of value between agents. An agent can be a person, an organization, a department or someone that is capable of controlling, acquiring, and providing resources (Andersson et al., 2005).

To Ryan Ko the process of business modeling involves essentially six phases, as shown in Figure 2.2 (Ko, 2009):

1. The first phase is the identification of business needs;
2. The second phase identifies the organization's mission. Management should present the high-level requirements to business analysts;

3. In the third phase, business processes are modeled using detailed diagrams. Typically, this stage uses a graphical language easy to understand, for example BPMN;
4. In the fourth phase the business processes models are translated to software or, in other words, the software that will support the business is developed. Usually a prototype is created.
5. At the fifth phase the experts evaluate the software and, if needed, add logical detail.
6. In the last phase the software that supports the business process is put into practice.

Several languages and tools have been proposed, and used, for the modeling of business processes. Most are based on graphical notations, such as data flow diagram, state transition diagrams, Petri nets, among others.

Next section is a survey of languages and tools for business process and software modeling.

2.5 Modeling Languages

This section presents the mains business process modeling languages and software modeling languages. The languages or models considered with more interest to the research are emphasized.

DFD - Data Flow Diagram

The Data Flow Diagrams (DFDs) were widely used in the information systems analysis, especially in the 70s and 80s of the twentieth century.

These diagrams can be used during the analysis phase allowing the users a better understanding of the system (Jalote, 2008). A DFD represents how the data flows within an organization. In this type of diagrams the system is seen as a function that transforms input data into the desired output data. A DFD provides a description of how the input data are processed to produce the output data (Jalote, 2008).

A DFD is composed by the following graphics components (Jalote, 2008):

- **Process:** represents a task that processes data or performs an action based on the data (Abi-Antoun et al., 2007). Graphically, a process is represented by a circle;
- **External Entities:** graphically represented by a rectangle, represents an organization or individual outside the scope of the system that sends or receives data that the system uses or produces.

- Data store: is graphically represented by two horizontal lines and represents a repository where the data is stored and retrieved.
- Data flows: graphically represented by an arrow, represents the data flow between the remaining elements: processes, data stores or external entities.

Besides the graphical elements the DFD also comprise a data dictionary. In the data dictionary, all components of the system are described.

In most cases it is not possible to describe with clarity, all data transformations that occur in a system in a single DFD (Jalote, 2008). To overcome these situations, the DFDs can be organized hierarchically at several levels. Usually the first diagram being drawn is the context diagram. In the context diagram, the whole system is represented as only one process with all its inputs, outputs and sources. This process is, then, refined (and detailed) in DFD level 0, which can give rise to various processes. Each of these processes can be expanded and detailed in a higher-level DFD and so on. The refinement stops if each process is considered to be “atomic” (Jalote, 2008). The detail level increases when a process in a higher-level DFD is refined into a lower-level DFD (Abi-Antoun et al., 2007).

ERD - Entity Relationship Diagram

The Entity Relationship Diagram (ERD) was created in the 70s of the last century, and it has been very well accepted since the beginning (Silingas and Butleris, 2008). Actually it is considered the default diagram used for planning and designing relational databases.

The ERD is a static view of the entities, their attributes and the relationships between them (Davis and Yen, 1999; Chen, 1976; Jalote, 2008).

An entity is something identifiable, or a concept in the real world that is important to the modeling purpose (Weske, 2012). The entities can be seen as types describing elements with common properties (Jalote, 2008). An entity is graphically represented by a rectangle and corresponds to a table in the database. The information, or the properties, about an entity are expressed through a set of attributes (Weske, 2012). These attributes can be instantiated into concrete values on the database. A relationship between two entities is represented through an association between those entities (Chen, 1976). The role of an entity in a relationship is the function that it executes in that relationship. Between two entities, we can distinguish three types of relationship (Chen, 1976):

- ($n : 1$) relationships: the existence of n elements in one side of the relationship is dependent on the existence of 1 element of the entity in the other side.

- ($m : n$) relationships: the existence of m elements in one side of the relationship may correspond to the existence of *1 or more* elements of the entity in the other side.
- (1 : 1) relationships: an univocal relationship is represented. The existence of 1 element in one side of the relationship corresponds to one and only one element of the entity in the other side.

The data model is often the basis for designing an application's database. Additionally, it may also be used to represent the business data structure (Weske, 2012), which is very important for the business processes management. However, the ERD does not specify how the data is manipulated or changed in the system (Jalote, 2008).

Domain Model

The domain model is used to structure the knowledge about a specific domain and is a way to leverage the elements (or concepts) of most interest on that domain (Evans, 2011). It represents the key concepts of the problem domain and the relationships between them. The key concepts are also called domain entities. Usually the entities attributes are also represented in the domain model. The Domain Model describes and constrains the scope of the problem domain.

Evans says that the domain model “is not just the knowledge in a domains expert's head, it is a rigorously organized and selective abstraction of that knowledge” (Evans, 2011).

According to (Jacobson et al., 1999) the domain model captures the most significant “things” belonging to the system scope. These “things” are represented as objects or classes and can represent business objects; real-world objects and events. The domain model aim is to understand and describe the most important classes within the system context (Jacobson et al., 1999). Usually the domain model is represented as a UML class diagram (Savié and da Silva, 2012).

UML - Unified Modeling Language

In 1997, the UML has emerged as a OMG (Object Management Group) standard and is currently considered as the modeling language used by default in companies (Olivé, 2010; Lange et al., 2006; Marshall, 1999; Bork and Fill, 2014).

According to Dobing & Parsons the use of UML in software development has “increased significantly and is now the second most widely used technique, after Entity-relationship diagrams” (Dobing and Parsons, 2009).

UML follows the object oriented paradigm and is more suited to model the technical side of information systems than the business side of an organization (Rittgen, 2008). UML is a graphical notation for expressing object-oriented designs (Fowler, 2004).

Some software development processes, such as UP (Unified Process), use the UML to support the modeling and documentation of the entire software development process (Jacobson et al., 1999; Marshall, 1999). A software development process is the set of activities needed to transform user requirements in a software system (Jacobson et al., 1999).

The UML 2.0 provides thirteen types of diagrams that allow the modeling of different points of view of the system and with different levels of abstraction (Lange et al., 2006; Yue et al., 2011). The UML allows to completely cover the entire software development cycle from requirements capture, through the use case diagram, to the way the various components fit together through the installation diagram (Pilone and Pitman, 2005; Sendall and Kozaczynski, 2003). Russell *et al.* Compare the UML to a “Swiss army knife” in terms of modeling and design (Russell et al., 2006). With its 13 different modeling notations (UML2.0), the UML allows representations from high abstraction level through the use case diagram, to the concrete representation of objects in the objects diagram.

UML diagrams can be grouped into two main categories (Yue et al., 2011): structure diagrams that are used to capture the physical organization of the system, and behavioral diagrams that focus on the behavior of system elements.

The diagrams are briefly described below:

- **Class Diagrams** - Class diagrams describe the system structure, i.e., identify existing entities, its internal structure and the relationships between them (Berardi et al., 2005; Yue et al., 2011). This type of diagram is one of the most widespread used UML diagrams.

The main graphical elements present in the class diagram are:

- Class - A definition of objects that share given structural or behavioral characteristics. Each class has attributes (a typed value attached to each instance of a classifier) and operations (a method or function that can be performed by instances of a classifier).
- Relationships - represents the relationships between classes. A relationship could have a name and can represent an association, aggregation, composition, generalization, etc..

A class diagram example is presented in Figure 2.3. This type of diagram is one of the most frequently used and well known UML diagrams. According to P. Fettke, UML class models describe the static structure of an Object-Oriented (OO) model and “are considered to be the most popular UML diagram used in practice” (Fettke, 2009).

- **Object diagrams** - These diagrams have the same syntax as the class diagrams and describe how instances of a class diagram are related at a given moment (Marshall, 1999).

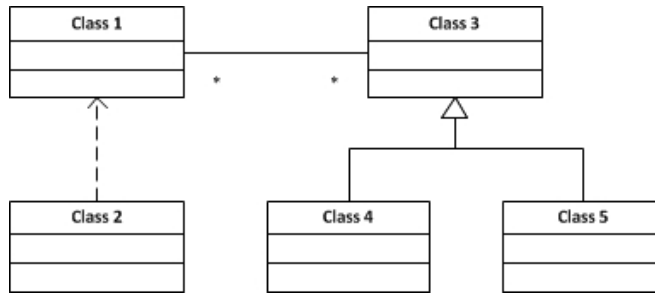


Figure 2.3: A Class diagram example

- **Package diagrams** - These diagrams are a particular type of class diagrams that focus on how the interfaces and classes are grouped together (Marshall, 1999).
- **Composite structure diagrams** - this type of diagrams appeared for the first time in UML2.0 in order to make the integration between the different existing class diagrams (Pilone and Pitman, 2005).
- **Components diagrams** - such diagrams describe the organization and dependencies of various software components (Jacobson et al., 1999). Can group small elements such as classes, or major elements such as sets of executable files.
- **Deployment diagrams** - this type of diagrams describe the various software components configuration (Pilone and Pitman, 2005), i.e., describes how the system runs and how each component is distributed through expected hardware.

The behavioral diagrams focus on the behavior of system elements. Among the behavior diagrams we have (Jacobson et al., 1999):

- **Use case diagrams** - use cases diagrams allow to describe the behavior of the system from the users point of view (Booch et al., 1998). This type of diagrams are detailed in section 2.5.
- **Activity diagrams** - These diagrams allow to capture the flow of activities in a process. Therefore, as can be read in section 2.5, these diagrams are often used to model business processes.
- **State machine diagrams** - this type of diagrams describe the sequence of states through which an element of the system passes. The elements may be small elements as a class, or the whole system (Gomma, 2011).

- **Interaction diagrams** - such diagrams are simplified views of the activity diagram in which, instead of emphasizing the activity performed, the elements involved in performing the activity are emphasized (Gomma, 2011). There are three types of iteration diagrams (Gomma, 2011):
 1. **Communication diagram** - is an interaction diagram that focuses on the messages exchanged by the elements involved in a particular behavior. The diagram gives particular attention to the objects involved rather than the order and nature of messages exchanged (Gomma, 2011).
 2. **Sequence diagrams** - this type of diagram emphasizes the type and order of messages exchanged between participants or objects during the execution. The sequence diagram captures the behavior (messages exchanged) of a single scenario (Fowler, 2004). The main graphical elements present in the sequence diagram are:
 - Lifeline - A vertical line that represents the sequence of events that occur in a participant during an interaction;
 - Actor - A participant that is external to the system in development;
 - Synchronous message - The sender waits for a response to a synchronous message before it continues;
 - Asynchronous message - A message that does not require a response before the sender continues;
 - Execution occurrence - A message that returns back to a participant that is waiting for the return from an earlier call.
 3. **Timing diagrams** - are a type of interaction diagrams that detail and specify the time of the messages. These diagrams are often used to show the change in state of an object over time in response to accepted events or stimuli (OMG, 2012). These diagrams are frequently used to model real-time systems since they allow you to specify how long the system has to process and respond to a particular message, events or stimuli.

The different diagrams give an insight of the system in different perspectives: the design perspective, development, implementation, process and use cases (Pillone and Pitman, 2005). But, as stated by Roussev, modeling the system in different perspectives and using successive refinements can give rise to consistency problems (Roussev, 2003). Thus, the different models must be consistent and compatible in order to the implementation become viable (Roussev, 2003). The study described in (Dobing and Parsons, 2006) concluded that the diagrams most commonly used are the class diagram, the use case diagram and the sequence diagram.

UML - Use Case Diagram

Use case models aim to capture and describe the functional requirements of a system (Hull et al., 2011; Yue et al., 2011). Booch *et al.* say that use case models, when defined by Ivar Jacobson, aimed to describe the behavior of the system from the users point of view (Booch et al., 1998). So, it is expected that a use case model specifies what a system is supposed to do (OMG, 2012). In (OMG, 2012), a use case is defined as a specification of a set of behaviors performed by an actor.

Among the thirteen diagrams provided by UML, the class model and the use case model are the most used (Dobing and Parsons, 2006). UML use case models are one of the most popular ways of capturing and describing the functional requirements of a system. This type of models have become even more popular with its inclusion in the Unified Process (also known as RUP) (Bittner and Spence, 2003a; Booch et al., 1998).

A use case model is a set of use case diagrams and the corresponding use case descriptions (Bittner and Spence, 2003a).

A use case model describes a sequence of interactions between the system and its users (Dijkman and Joosten, 2002a; Cockburn, 2001). The use case model is used in requirements elicitation and specification as a means to facilitate the dialog with the customer about what the system is supposed to do (OMG, 2012).

Whittle and Jayaraman (Whittle and Jayaraman, 2006) define use cases as a set of scenarios where a scenario is “an expected execution trace of a system”. In fact, one possible approach to model a system, using a use case model, starts by identifying all possible scenarios and then generalizes them in order to create the use case model (Issa, 2007).

According to A. Cockburn “a use case is a description of the possible sequences of interactions between the system under discussion and its external actors, related to a particular goal” (Cockburn, 2001).

A use case model should identify the system boundaries (marked by a rectangle) and identify the actors which are represented by a “stickman” icon outside the system boundaries (Hull et al., 2011; OMG, 2012). An actor is someone or something that interacts with the system (OMG, 2012). So, an actor is always related to one or more use cases. A use case is graphically represented by an ellipse and contains a brief description of the action (Bittner and Spence, 2003a) as can be seen in Figure 2.4.

A use case diagram is composed by actors and use cases. Each use case shall have an associated description and can have pre-conditions and post-conditions. There are some alternatives that can be used to describe a use case, like informal text, numbered steps, pseudo-code, among others (Machado et al., 2005).

Use cases can be related through *include* and *extend* relationships. A use case can *include* another use case and can be *extended* by other use cases (Berenbach, 2004).

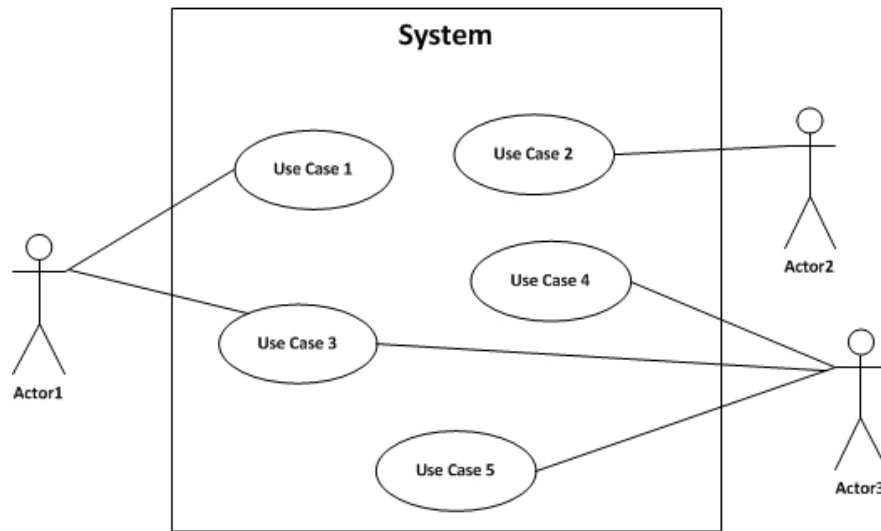


Figure 2.4: A UML use case diagram example

UML - Activity diagrams

UML is a modeling language widely disclosed and actually used in companies, particularly in software development companies (Dobing and Parsons, 2006). But despite UML being traditionally used to model software applications, its flexibility and extensibility attracted modelers and business analysts. Thus, the UML activity diagram (AD) has also been used to describe the behavior of business processes. The sequence diagram can also be used to specify, at a lower level, the interactions between the participants (Dorn et al., 2009).

Within the set of diagrams provided by UML, the activity diagram is named as the most important for business modeling (Dijkman and Joosten, 2002a). The activity diagram can show the activities in sequence or in parallel, the objects produced or consumed by an activity, who is responsible for the activity and the relationship, or dependencies, between activities. All this is essential in the business process modeling (Eriksson and Penker, 2000), which as we saw earlier, is a set of related activities designed to produce a result (product or service).

The activity diagram (Linzhang et al., 2004) represents mainly the following concepts:

- actions - atomic executions, not interruptible with irrelevant runtime;
- Activities - ongoing non-atomic execution, interruptible, where the execution time is usually relevant;
- Transitions - represent the control flow between two activities;
- Objects - Can be divided in two types of objects: object flow and object state;

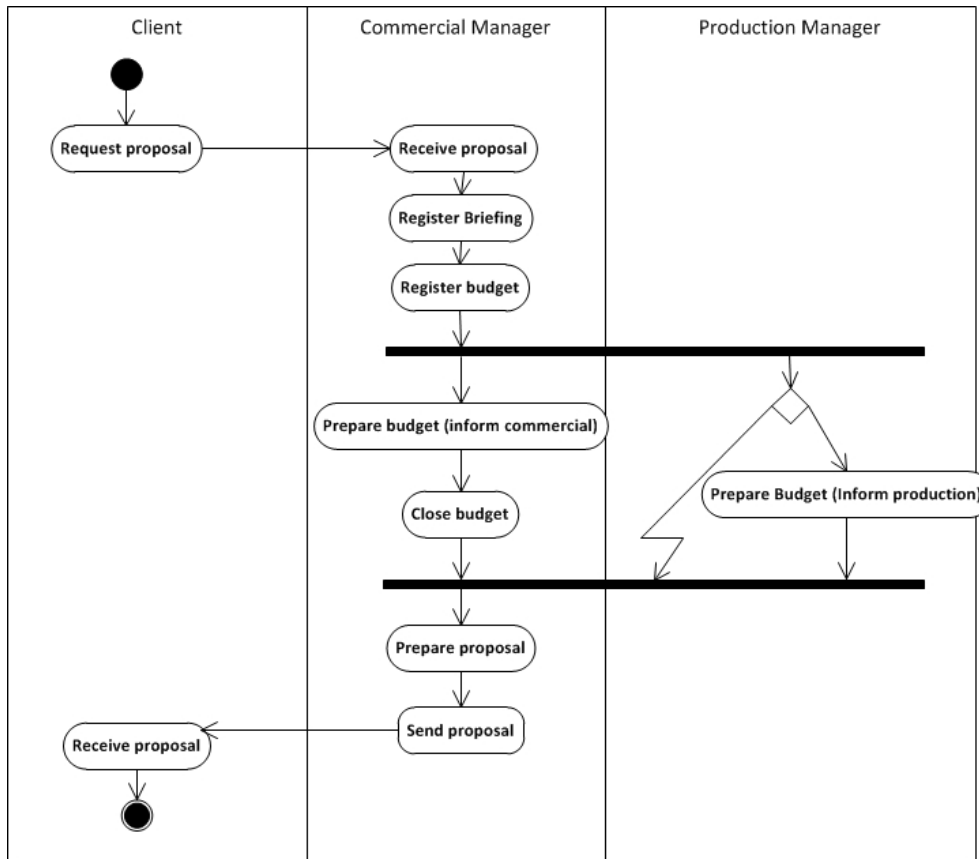


Figure 2.5: Activity Diagram example (adapted from (Silva and Videira, 2005))

- Decision points and junction points;
- Transitions - diffusion points (*fork*) and joint or synchronization;
- swimlanes - a partition for organizing responsibilities for activities.

The activities are graphically indistinguishable from actions and are represented by a rectangle with rounded corners. Transitions are represented by solid arrows. The fork and join are represented by a solid line. The initial state is represented by a solid ball and the final state by a ball within a full circle.

The activity diagram also provides the concept of partitions (or *swimlanes*) as an element that allows grouping activities that are from a participant responsibility. Each group is separated by a dashed line.

In Figure 2.5, an example is presented of an activity diagram with *swimlanes*, of a “prepare proposal” business process.

Despite its usefulness and clarity, certain aspects of business processes are not mappable to activity diagrams. However, the standard UML has concepts as stereotypes, tagged values and restrictions, which allow to create extensions

relatively easily. For these reasons, several extensions have been proposed, of which three are discussed in the following subsections.

UML - Eriksson and Penker extension

Eriksson and Penker claim that there are several similarities between the business modeling and software modeling. However there are also some differences. The business management involves factors such as people, equipment, rules, among others, which are not transformable into executable programs. For this reason, Eriksson and Penker propose an extension to UML in order to more clearly cover and visualize important concepts for business process objectives, resources and business rules (Eriksson and Penker, 2000).

Eriksson and Penker use stereotypes, tagged values and constraints as base to the proposed extension. A business process can have the following tagged values: objectives, goals, resources, products, events, order in which the activities are performed, among others.

The Eriksson and Penker extension uses five models for business description (Eriksson and Penker, 2000):

- Conceptual model - uses the UML class diagram for the identification and definition of the business key concepts;
- Goal/problem model - uses the UML object diagram to represent the goals or business objectives to be achieved;
- Resources model- represented by a class diagram, identifies the resources (or product information) involved in the business;
- Information model - represented by a UML class diagram or by an object diagram, is a particular case of the resource model and represents the information in a structured manner to facilitate its understanding;
- Organizational model - represented by the a class diagram or by an object diagram, represents the organizational structures of business.

Eriksson and Penker, also propose a new set of diagrams, such as assembly line diagram, process diagram (represented by UML activity diagram), goal-problem diagram, etc.. To this end, the authors also propose a set of new graphical objects, some of them can be seen in Figure 2.6 which presents an example of a process diagram based on the UML activity diagram and applied to the *prepare a proposal* business process for a web site development.

With the proposed models and diagrams, the Eriksson and Penker extension allows to fully model a business process.

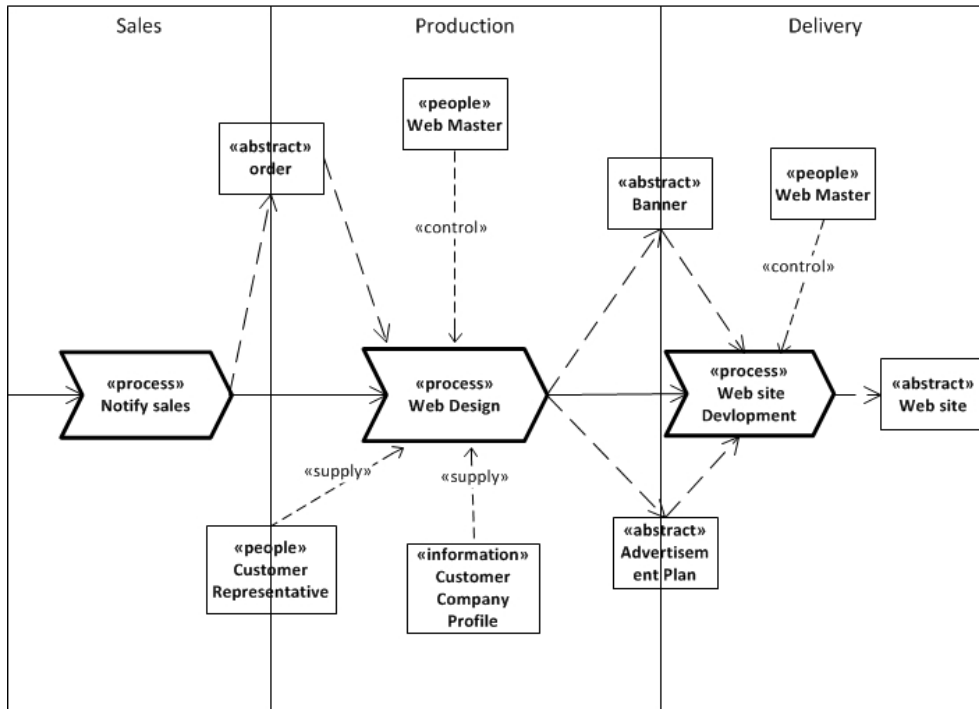


Figure 2.6: Process diagram example (adapted from (Eriksson and Penker, 2000))

UMM - UN/CEFACT

UMM (UN/CEFACT's Modeling Methodology)¹ is a business modeling methodology based on standard UML which includes three different model views (Dorn et al., 2009; Hofreiter et al., 2010):

- The business domain view (BDV) - used to collect the knowledge from stakeholders;
- The business requirements view (BRV) - used to capture the requirements for exchanging business documents between organizations;
- The business transaction view (BTV) - is used to express the global exchanges of document between the potential business partners.

The UMM was especially designed to support the modeling of B2B (Business-to-Business) processes (Hofreiter et al., 2010). The UMM allows to capture the semantics of the business in terms of pre-defined templates. This approach allows us to differentiate the definition of web services from their technological implementation (Karakostas et al., 2006). The UMM also allows to model the business data exchanged in a business process (Hofreiter et al., 2010).

¹UN/CEFACT - United Nation's Centre for Trade Facilitation and Electronic Business

UMLEWM - UML Extended Workflow Metamodel

The UML activity diagram allows the modeling of business processes, however, the level of detail is not enough to support the detail level demanded by a workflow process (Debnath et al., 2006). Thus UMLEWM (UML Extended Workflow Metamodel) incorporate the ability to model the workflow process by extending UML meta-model with new meta-classes. The new meta-classes represent elements present in the workflow meta-model and absent in the UML meta-model.

BPMN - Business Process Model and Notation

The BPMN language was originally developed by a software companies consortium, the BPMI (Business Process Management Initiative) (Allweyer, 2010). The development of the first version, published in 2004, was led by Stephen A. White from IBM Company. By that time, the BPMI became part of the OMG. The OMG is the organization responsible for developing various software standards such as UML and CORBA (Common Object Request Broker Architecture). In 2006, the BPMN was officially accepted as a standard of OMG (Allweyer, 2010). Version 2.0 was completed into early 2011 (OMG, 2011a).

BPMN is a graphical language for modeling business processes (Lam, 2009) that calls itself the “lingua franca” to document, visualize, specify and model the business process. This language provides a set of graphical symbols that facilitates the construction of business process model (Liang et al., 2008). For Dorn *et al.*, BPMN provides a small but clear notation for modeling business process (Dorn et al., 2009).

The BPMN modeling language was developed with the aim of providing a notation understandable by all stakeholders involved in the business process (OMG, 2011a). BPMN 2.0 provides three main types of diagrams (OMG, 2011a):

- **Process diagrams** - Define a set of business activities carried out by an organization for the concretization of a goal (product or service). The business process includes the flow and use of information and resources.
- **Choreography diagrams** - This is a type of process diagram that describes how participants coordinate their interaction. Such diagrams can be used to analyze how the participants exchange information in order to coordinate their actions, as we can see in the example in Figure 2.7. A choreography defines a set of interactions between the participants. A participant, in general, defines a role in the organization or a business partner and is represented in a pool. A Pool represents a participant or a group of participants in the process and graphically is represented by a rectangle. Thus, choreography involves at least two pools. The exchanged information is represented by the incoming and outgoing messages.

This diagram can be seen as the business contract between two or more organizations, or business partners.

- **Collaboration diagrams** - This type of diagram focuses on the exchange of information between participants, represented by pools. The collaborations are modeled with two or more pools, each containing a separate process. The processes communicate by exchanging messages (Allweyer, 2010). Collaborations are permitted in all combinations of pools, processes and choreographies.

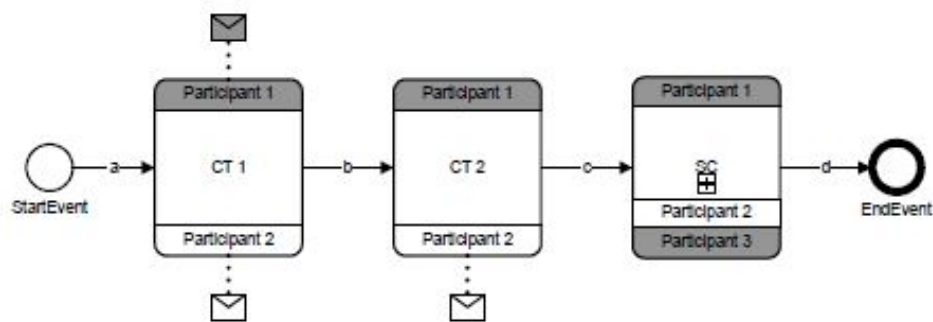


Figure 2.7: A BPMN choreography example (extracted from (OMG, 2011a))

The basic process models can be grouped into two types of processes (OMG, 2011a):

- **Private Business Processes** - A private process is a process internal to a specific organization. Each private process is represented within a Pool. The process flow must be in one pool and should never cross the boundaries of that Pool. The interaction between distinct private business processes can be represented by incoming and outgoing messages. The message flow can cross the pool boundaries.
- **Public Processes** - A public process represents the interactions between a private business process and other processes or participants. Only activities that are used to communicate with the other participants must be included in the public process. Thus, a public process shows to the outside world the origin of the messages needed to interact with that process and the messages flow order, but can keep private the remaining activities.

Public processes can be modeled separately or within a Collaboration to show the flow of messages between participants and external process activities.

The BPMN captures business processes by defining the Business Process Diagrams (BPD). The notation used by BPMN diagrams became influenced by other

notations such as UML activity diagrams, IDEF, EPCS, among others (OMG, 2011a).

The business process diagrams use a set of graphical objects that can be grouped into five basic categories (OMG, 2011a):

- **Flow Objects** - are the main graphical elements to define the behavior of a business process. There are three kinds of Flow Objects: Events, Activities and Gateways.
- **Data** - represent the data involved in a business process.
- **Connecting Objects** - model the connection between the several process elements. There are four types of connecting objects: Sequence Flows, Message Flows, Associations and Data Associations.
- **Swimlanes** - represent the participants in the process. A participant is a person, or something, involved in the process. Participants in the process can be grouped into pools or, more particularly, in Lanes. A pool can be divided into several Lanes, for example, to represent the different departments of an organization involved in the process. A Lane is a sub-partition of a pool, so it is always within a Pool.
- **Artifacts** - are used to provide additional information to the process, such as a note (“Text Annotation”). The artifacts can be used to define possible extensions to the BPMN language.

The following subsection addresses data in BPMN 2.0, mainly its representation and flow.

Data in BPMN

Process modeling must be able to model the data items (physical documents or electronic information) that are created, manipulated, and used during the execution of a process (OMG, 2011a). The data involved in the process can be considered persistent or not persistent (volatile). The persistent data is the one that remains beyond the life cycle, or the scope, of the process (OMG, 2011a).

In BPMN 2.0 the data can be represented in a process diagram by the elements presented in Figure 2.8. Data manipulation elements can be grouped into:

- **Data Objects** - data objects represent the information that flows through a process. A data object can be referenced by `DataObjectReference`. A data object reference is a way to reuse one data object in the same diagram. A data object reference can represent a different state of the same data object at different points in the process. On a process diagram this is represented by (OMG, 2011a) `< DataObjectName > [< DataObjectReferenceState >]`.








	Name	Symbol	Description
Data Objects	Data Object		Data objects represent the information needed or produced by the activity. A Data object can be referenced by DataObjectReference
	Data Object Collection		Represents a data objects' collection. (Attribute <code>isCollection</code> : boolean = true)
	Data Input		Represents the information needed for the process execution
	Data Input Collection (input set)		Represents a data input collection (Attribute <code>isCollection</code> : boolean = true)
	Data Output		Represents the information produced by the process execution
	Data Output Collection (output set)		Represents a data output collection. (Attribute <code>isCollection</code> : boolean = true)
	Data Store		Represents the persistent data stored or retrieved by one activity. A Data Store can be referenced by DataStoreReference

Figure 2.8: Data representation elements (adapted from (OMG, 2011a))

- **Data Object Collection** - represents a set of data objects.
- **Data Inputs** - data external to the process that can be read or received by an activity.
- **Input Set** - represents the data needed for the process to work properly.
- **Data Outputs** - data available as a result of a process activity.
- **Output Set** - represents the information produced by the process and exported abroad (Allweyer, 2010).
- **Data Store** - a data store is a means to handle persistent data. It provides a mechanism for an activity to store information or use the information stored. A data store can be referenced by `DataStoreReference` which can be used to represent the same data store at different points in the process.

A data object can represent any type of information, such as electronic data, documents, forms or other physical data (Allweyer, 2010; List and Korherr, 2006). A data store can represent paper documents (a file folder, an agenda, a notebook, etc.) or an electronic database. Data objects and data stores are exclusively used in process diagrams (OMG, 2011a).

The next subsection describes the data flow representation.

Data Flow

During the process execution, resources and/or data are consumed and produced. The transmission of the data created or used during a process execution can be represented by:

- **Messages** - A Message is used to represent the contents of a communication between two participants. Each participant is represented by a different pool. So, a message crosses the pool boundary to show the interactions between separate private business processes (OMG, 2011a). A message can represent any kind of information like an email, a fax, a letter, a phone call, etc. (Allweyer, 2010). Graphically, an initiating message is represented by a white envelope. An non-initiating message is represented by a gray envelope (OMG, 2011a). A message can only be transmitted between different pools. It is not allowed to use messages within the same pool.
- **Data Associations** - A Data Association can be used to model how data comes and goes from activities or events. This way it is possible to identify the activity that produces a certain data object and the activity that uses a given data object. This also allows the possibility to identify the activity that sends data to a data store and the activity that gets data from a data store. A Data Association can only be used within a pool. Data associations can be divided into two types:
 - Data Input Association - represents the input data.
 - Data Output Association - represents the output data.

A data store can also be considered as a mean for transmitting information when the information written by an activity is read by another. In this situation, all the activities involved must have access to the same data store, so, they should be represented within the same Pool if we are dealing with activities belonging to the same process (Allweyer, 2010).

Kocbek *et al.* concluded that the BPMN is being increasingly used and is consolidating the position of “de facto standard in the business process modelling field” (Kocbek et al., 2015).

It is strongly recommend the use of modelling tools to create BPMN models allowing preventing errors by enhancing the syntax validation. Currently there

are several tools on the market that enable the creation and verification of syntactic correctness of designed models, as well as export/import these models. Examples of these tools are ECLIPSE - Model Development Tools (MDT), the Modelio - Modeling and Implementing software and systems ², WebRatio BPM Free³, Bizagi process modeler⁴, among others.

Petri-nets

The Petri nets, which name is due to the German mathematician Carl Adam Petri, are graphical mathematical representations, traditionally used for the systems representation and modeling. Currently they may also be used for modeling business processes (Giaglis, 2001).

According to Dong and Chen, Petri nets are one of the most popular and powerful models to represent the system analysis that allow competition, parallelism, non-determinism and sharing resources (Dong and Chen, 2005).

The authors suggest three reasons for the use of Petri nets:

- Include formal semantics;
- They are based in states;
- They have an abundance of analysis techniques.

Petri nets are graphs composed of a set of places and a set of transitions. The places (or states) are usually represented by circles and transitions by rectangles (or bars). The states are connected to the transitions through directed arcs, as we can see in the simple Petri net example in Figure 2.9.

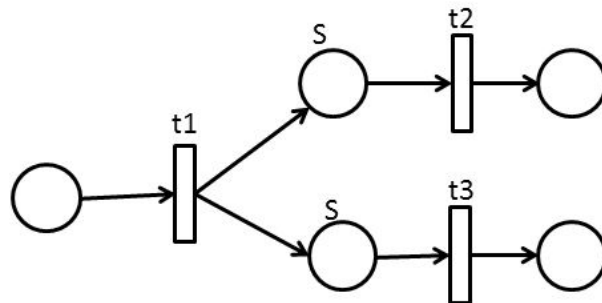


Figure 2.9: A Petri-net example (adapted from (Dong and Chen, 2005))

A Petri net may be defined as a tuple $R = (P, T, AE, AS)$, where:

²www.modelio.com

³www.webratio.com

⁴www.bizagi.com

1. $P = \{ p_1, p_2, \dots, p_m \}$ is a finite set of places;
2. $T = \{ t_1, t_2, \dots, t_n \}$ is a finite set of transitions. The sets L and T are disjoint;
3. AE : is a set of arcs incoming for transitions;
4. AS : is a set of arcs outgoing of the transitions;

Dijkman *et al.* say that Petri nets are particularly suited to model a system behavior in terms of flow. It could refer to the flow of control or flow of objects or information (Dijkman et al., 2008).

According to Giaglis, Petri nets are not sufficiently succinct to be used in processes modeling of high-level complex business, so several Petri nets extensions have been studied and developed (Giaglis, 2001), such as High-Level Petri nets (Buscemi and Sassone, 2001; Eike et al., 1998), colored Petri nets (Giaglis, 2001), hierarchical Petri nets (Oswald et al., 1990), Time Petri nets (Ling and Schmidt, 2000), etc..

High-level Petri nets allow a more compact specification thus making possible its application to more complex systems (Giaglis, 2001; Buscemi and Sassone, 2001).

The colored Petri nets contain information on the tokens, represented by colors (Giaglis, 2001). The transitions are labeled with expressions which evaluate the set of colors after each color is assigned to a variable. An input arch symbolizes the need for resources and a output arc symbolizes the creation of resources. Resources are graphically represented by small black circles within the places to which we give the name brand.

The time Petri nets allow the transitions to “hold” token for a given range of times. This type of Petri nets allows to model interaction between activities taking into account their start and end times and allows modeling of shared resources available at different times (Ling and Schmidt, 2000).

Little-JIL

Little-JIL is a high-level programming language, with a formal syntax and a rigorous operational semantics. This language is executable, and can be used to coordinate processes (Cass et al., 2000).

According to Lerner *et al.*, Little-JIL uses a clear and precise graphical notation, which supports the processes definition (Lerner et al., 2010).

Little-JIL is a language to coordinate agents. The program describes the order and communication made between the various units of work, called steps. The step can be associated with an agent (Group, 2006). An agent in this context is an independent entity that specializes part of the process. An agent is responsible for performing a job and report the success or failure when it ends.

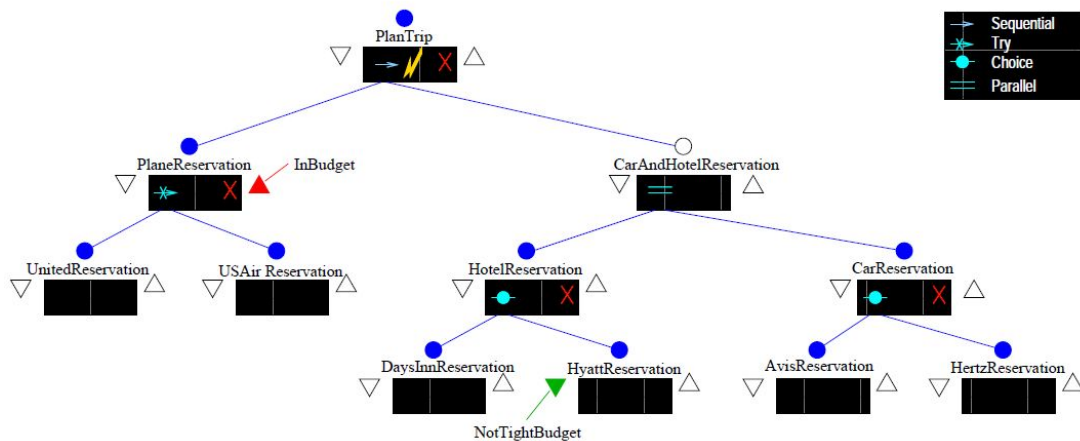


Figure 2.10: Example of a Little-JIL diagram (extracted from (Wise et al., 2000))

An agent may be human or software, the Little-JIL does not distinguish them. Both human and software agents have an agenda. What distinguishes them is how each type of agent connects to the agenda. Thus human agents use a GUI (Grafical User Interface), which interacts as interpreter for the API (Application Programming Interface) as software agents are connected directly to the API.

Little-JIL is a language easy to use (Wise et al., 2000), understandable by non-programmers, and supports the coordination of complex processes of software engineering. A coordination language must support a wide variety of process abstractions such as organizations, activities, artifacts, resources, events, agents and exceptions, which can easily make a large and complex language.

In Little-JIL a step is the central point of coordination and provides a mechanism for scoping, control data flow, exceptions and resource allocation to agents. The steps are organized in a hierarchical static manner, but can have a highly dynamic execution structure including the possibility of recursion and concurrency (Cass et al., 2000).

A program in Little-JIL is a tree constituted by various kinds of steps, each one could be multi-instantiated at runtime. The leaves represent the smallest unit of specific work. The structure of the tree represents the way in which this work will be coordinated (Cass et al., 2000), as shown in Figure 2.10.

The activities coordination involves (Cass et al., 2000):

- A group of agents, each capable of performing one or more tasks in support of the activity;
- A communication mechanism that allows the sharing of information among agents;
- A distribution mechanism that allows agents to operate on separate machines;

- A process of allocating tasks among agents;
- A coordination process that links the agents to tasks in a favorable manner to the activity.

The language has a number of features that allows specifying the program and coordinating the process steps. The six main characteristics are described below (Wise et al., 1999):

1. Four different types of steps: sequential, parallel, try and choice (see Figure 2.10);
2. Requirements: are mechanisms to add checks before (∇) and after (Δ) a step being executed, to verify if all the necessary conditions to start the step are satisfied and if the step was correctly performed;
3. Exceptions and handlers: are flow control arguments. Are used to indicate and correct errors or exceptional conditions that may occur during the execution of the program;
4. Messages and responses: identical to exceptions, the difference is that it is not possible to propagate messages by trees, unlike exceptions. Messages have a global scope;
5. When passed between steps allow the transmission of the information. A step can receive information by a parameter and returns the execution results a step being executed;
6. Resources: are represented as entities that are required during the step execution.

The Little-JIL provides support for complex processes coordination and is a language easy to use and understandable by non-programmers. For those reasons its use and disclosure tends to increase.

EPCs - Event-driven Process Chains

Event-driven Process Chains (EPCs) is a graphical language, intuitive, used to describe the business process. This language is more focused on the logic level of the business process (van der Aalst, 1999).

An EPC model is based on a sequence of events and activities (functions) that constitute the business process. The logical connectors (*and*, *or* and *xor*) allow the description of the branching actions and conditions to execute the activities in parallel (van der Aalst, 1999). In these models, actors that perform activities are represented by ellipses. The activities executed by an actor are placed in his/her

respective swimlane and are represented by rectangles. Events are represented by hexagons (Cardoso et al., 2009).

EPCs are process oriented modeling techniques and are used to define processes in SAP/R3 and other ERP systems (Dorn et al., 2009). EPCs are supported by the ARIS toolset (Dorn et al., 2009; van der Aalst, 1999).

IDEF - Integrated Definition Methods

The IDEF (Integrated Definition for Function Modelling) is a modeling techniques family developed as a set of notational formalisms to represent and model processes and data structures (Giaglis, 2001).

To Aguilar-Saven, IDEF is a family of methods that supports a paradigm capable of dealing with the modeling needs of a company and its business areas (Aguilar-Savén, 2004).

Dorn *et al.* claims that IDEF is a set of modeling languages used especially by the U.S. government (Dorn et al., 2009). Each IDEF family member is used according to different applications. The most important are: IDEF0, IDEF1, IDEF1X, IDEF2, IDEF3, IDEF4 and IDEF5. To the business process modeling the most suitable are IDEF0 and IDEF3 (Aguilar-Savén, 2004). Next, the main IDEF family members are described according to (Aguilar-Savén, 2004; Dorn et al., 2009; Giaglis, 2001):

- The IDEF0 was designed to model decisions, actions and activities of an organization and its goal is to model the functional perspective. The model represents the activities of high-level processes and indicates their inputs and control outputs. These models are composed of three kinds of information, graphic diagrams, text and glossary. The main component is the graphical diagram, but the three components are connected.
- The IDEF3 was developed to overcome some limitations of IDEF0. The IDEF3 describes the process as an ordered sequence of activities or events. This model uses two complementary diagrammatic representations of the process model: process flow diagram that describes the flow of process activities and the state transition diagram of an object, which represents how the different states of an entity flows in a process. IDEF3 is suitable both to modeling small processes and complex processes.

IDEF3 is also used in various other areas such as the Business Process Engineering (BPE), the Business Process Reengineering (BPR) and in the definition, development and software maintenance.

- The IDEF1 is used to model information.

- The IDEF1x was designed as a technique for modeling and analyzing data structures for defining the requirements of information systems. It is based on the entities and relationships model (ERD).
- The IDEF2 is a method for designing simulation models and is used to represent the behavior, over time, of the resources involved in a production system.
- The IDEF4 is a method of object-oriented design, developed to support the object orientation paradigm.
- The IDEF5 provides a method theoretically and empirically well grounded, specifically designed to assist in the creation, modification and maintenance of ontologies⁵.

Since they belong to the same family of techniques, IDEF models complement each other and when combined can provide a full perspective of the modeling system. However, the development and maintenance of this set of different models can be complex and time consuming.

XPDL - XML Process Definition Language

O XPDL (XML Process Definition Language) is a language based on XML (Extensible Markup Language)⁶. XPDL was proposed by WfMC to exchange process definitions between different products workflows (van der Aalst, 2003; Coalition, 2011).

XPDL is a standard for exchange of business process definitions between different products (Dorn et al., 2009). The purpose of this language is to provide a “lingua franca” for the workflow domain, allowing import and export process definitions between different modeling tools and management systems of workflows (van der Aalst, 2003).

The main XPDL elements are (van der Aalst, 2003): Package, Application, Workflow-Process, Activity, Transition, Participant, DataField and DataType. A simple example of a XPDL file can be seen in Figure 2.11.

The XPDL has been widely used to support simulation. This is mainly, because the XPDL package is complemented by a number of additional schemes which provide additional information necessary for the simulation, including details about the performance of activities (e.g., information on life), arrival times of work, the characteristics of the resources (cost, etc.) and a variety of simulation options.

The XPDL provides a standard format for files to support the exchange (import/export) of BPMN diagrams. The XPDL does not typically serves to model

⁵Ontology is a part of philosophy whose goal is to divide the “world” among different objects.

⁶XML is considered a standard for data exchange

```

1 <WorkflowProcess Id="Sequence">
2   <ProcessHeader DurationUnit="Y"/>
3   <Activities>
4     <Activity Id="A">
5       ...
6     </Activity>
7     <Activity Id="B">
8       ...
9     </Activity>
10  </Activities>
11  <Transitions>
12    <Transition Id="AB" From="A" To="B"/>
13  </Transitions>
14 </WorkflowProcess>

```

Figure 2.11: A XPDL file excerpt (extracted from (van der Aalst, 2003))

processes, but can be used to export BPMN processes, providing capacity for portability among various tools.

BPEL - Business Process Execution Language

BPEL (Business Process Execution Language) is essentially an extension to imperative programming languages with constructs related to the implementation of process-oriented web services. BPEL is a language based on XML (Ouyang et al., 2006) that supports structured programming instructions as *if*, *while*, among others, which allow the execution of instructions in sequence and in parallel. Once its focus is process-based business services, BPEL provides support for sending messages.

BPEL combines features from IBM’s WSFL (Web Services Flow Language) and Microsoft’s XLANG (Web Services for Business Process Design) (Bichier and Lin, 2006).

A BPEL process definition comprises two categories of activities: basic activities and structured activities. Basic activities correspond to atomic actions, such as a service invocation, a message reception, reply to a request, assign a value to a variable, or terminate a process. Structured activities are related to the behavioral constraints imposed on the implementation of a set of activities, such as the definition of an execution order, the execution flow in parallel, conditional execution, or group of activities in a block.

Throughout the times new versions of the BPEL were created. Thus, the version BPEL4WS (Business Process Execution Language for Web Services) was widely used until the appearing of the latest version, the WS-BPEL 2.0 standard. The two versions are incompatible at several points. Hofstede *et al.* state that BPEL4WS is a web services composition languages that can be used to “glue” services previously defined (ter Hofstede et al., 2003).

The WS-BPEL, originally introduced by IBM, Microsoft and BEA, is an executable language for specifying the business processes behavior based on Web services, which can create a new Web service from existing ones (Liang et al., 2008).

To describe complex business processes, BPEL defines several activities including the invocation of services, operations on data, among others. The process is defined based on the interaction between partners. A process may require a service, can provide a service or can interact in both directions.

A simple example of a BPEL file can be seen in Figure 2.12.

```

<BPELSuitcase>
  <BPELProcess src="Travel.bpel" id="TravelProcessCh4">
    <partnerLinkBindings>
      <partnerLinkBinding name="client">
        <property name="wsdlLocation">
          Travel.wsdl
        </property>
      </partnerLinkBinding>
      <partnerLinkBinding name="employeeTravelStatus">
        <property name="wsdlLocation">
          http://localhost:9700/orabpel/default/Employee/Employee?wsdl
        </property>
      </partnerLinkBinding>
      <partnerLinkBinding name="AmericanAirlines">
        <property name="wsdlLocation">
          http://localhost:9700/orabpel/default/AmericanAirline/AmericanAirline?wsdl
        </property>
      </partnerLinkBinding>
      <partnerLinkBinding name="DeltaAirlines">
        <property name="wsdlLocation">
          http://localhost:9700/orabpel/default/DeltaAirline/DeltaAirline?wsdl
        </property>
      </partnerLinkBinding>
    </partnerLinkBindings>
  </BPELProcess>
</BPELSuitcase>

```

Figure 2.12: A BPEL file excerpt (extracted from (Juric, 2015))

For Liang *et al.*, BPEL is an executable language for specifying business processes that are not easily understood by business stakeholders (Liang et al., 2008). BPEL provides a way to model the behavior of both executable processes and abstract processes (Bichier and Lin, 2006).

According to M. Juric, BPEL (or BPEL4WS) enables the realization of service oriented architecture through “composition, orchestration, and coordination of web services” (Juric, 2015).

SPEM - Software & Systems Process Engineering Meta-Model Specification

SPEM (Software & Systems Process Engineering Meta-Model Specification) was defined by OMG for developing modeling processes (OMG, 2008). The SPEM 2.0 Meta-Model is a MOF-based model (OMG, 2008). A meta-model is a model that represents the language used by other models (Bézivin, 2006). The MOF (Meta Object Facility) is an OMG standard for the definition of the abstract syntax of modeling languages (meta-models) (OMG, 2011c).

Debnath *et al.* claim that, in the context of SPEM, a meta-model is used as a language to describe a concrete software development process or family of related software development processes (Debnath et al., 2006).

According to (OMG, 2008), “SPEM 2.0 is used to define software and systems development processes and their components”.

The SPEM aim is to support the definition of software development processes, including processes that involve the use of UML, and RUP. In SPEM, a software development process is the collaboration between the roles involved in the process, activities and work products.

The SPEM is basically formed by two groups of packages: the *SPEM Foundation* and the *SPEM Extensions*. The *SPEM Foundation* is a subset of UML 1.4 which describes the static structure of the model. The *SPEM Extensions* adds constructors and semantics required by the engineering of software processes (Debnath et al., 2006).

According to (OMG, 2008), SPEM 2.0 meta-model is structured into seven main meta-model packages:

- Core: this package contains the set of classes present in all other meta-model packages.
- Process Structure: this package defines the base for all process models, supporting the creation of simple and flexible process models.
- Process Behavior: allows extending process structure with behavioral models, providing “links” to existing externally-defined behavior models.
- Managed Content: Contains concepts for managing the textual content of natural language descriptions used in development processes.
- Method Content: provides the concepts for SPEM 2.0 users and organizations to build up a development knowledge base that is independent of any specific processes and development projects.
- Process With Methods: this package defines new and redefines existing structures for the integration process.

- Method Plugin: This package introduces concepts for the design and management of libraries and configurable content repositories and processes method.

2.6 Final Remarks

All the languages referred to above have equivalent aspects and to some extent they complement each other. Several authors present studies evidencing this observations, some of which are considered of more interest in the scope of this work, and are presented next:

- Debnath *et al.* presents an approach that maps between SPEM and UML-EWM (Debnath et al., 2006).
- Dijkman *et al.* studied the equivalence between BPMN and Petri nets in (Dijkman et al., 2008).
- Liang *et al.* propose a method for mapping the changes made in BPMN models to BPEL (Liang et al., 2008).
- In (Raedts et al., 2007) the authors present the semantic and behavioral equivalence between BPMN models and Petri nets.
- In (Lam, 2009) V. Lam proves that two different graphical representations of business processes are behaviorally equivalent.
- Ouyang *et al.* present a technique to generate BPEL from a subset of BPMN models and from UML (Ouyang et al., 2006).

Other authors extend the existing notations, adapting them to particular cases. Others create new tools, such as the example of (Kees van Hee et al., 2006). The Yasper is a tool for modeling, simulating and analyzing systems workflow based on Petri nets.

A business process modeling language should be able to provide information to their users on elements such as: what activities constitute a process, who performs those activities, when and where those activities should be performed and what data are involved. Thus, to provide all the necessary information, the modeling language should be able to represent one or more of the following perspectives (Giaglis, 2001):

- Functional - represents **which** activities (process elements) can be performed;
- Behavioral - represents **when** and **how** the activities should be performed;

Language	Functional	Behavioral	Organizational	Informational
BPMN 2.0	Yes	Yes	Yes	Yes
Petri nets	Yes	Yes	No	No
Activity diagram (UML)	Yes	Yes	No	No
Eriksson-Penker extension	Yes	Yes	Yes	Yes
IDEF	Yes	Yes	Yes	Yes
EPCs	Yes	Yes	Yes	No
Little-Jil	Yes	Yes	No	No

Table 2.1: Different perspectives of business processes modeling languages (based on (Giaglis, 2001))

- Organizational - represents **where** and **by whom** the activities should be executed, the physical communication mechanisms used to transfer entities and physical locations and means for storing;
- Informational - represents the **data** (informational entity) handled by a process and their inter-relationships.

The different studied languages and approaches have different ways of providing information on the different perspectives, as we can see in Table 2.1. Looking to the Table 2.1 we may conclude that the BPMN, IDEF family, Eriksson and Penker extension languages are more complete than the others. However, the language more appropriate to a problem resolution depends on the characteristics of the system being modeled.

Some languages can be used either for business process modeling as well as for software modeling, however, some are considered better suited to model business processes and others to model software.

The BPMN language, besides being a complete language, is a standard, easy to learn and to use, and is considered as a language used by default in business process modeling (Kocbek et al., 2015). For this reasons the BPMN language is used in this research work.

The next chapter addresses existing approaches that make the relationship, or conversion between different models.

Chapter 3

Deriving Models in Information Systems Development

In this chapter we will focus on existing approaches that make the linkage between software models and business process models, more specifically on approaches that approximate the software development area with the business process modelling area.

3.1 Introduction

A business process model identifies the activities, resources and data involved in the creation of a product or service, integrating several useful information that can be used to create software models for the supporting software system.

It is recognized that the software that supports the business must be aligned with the business processes. Therefore, it is natural to try an approximation between business process modeling and software modeling.

Following this idea, several authors proposed approaches to derive software models based in business process models. Some of the created approaches are presented next in this chapter.

We start by presenting the approaches that, in general, attempt to go from the business process models to software models (section 3.2). Section 3.3 addresses approaches that try to get the data model based on business process models. Then we will focus on approaches which goal is to obtain the use case diagrams based on the business process models (section 3.4). Approaches that try to obtain the class diagrams, most of them based on use case models, are addressed in section 3.5. Approaches to decompose and refine use case models are presented in section 3.6. Section 3.7 presents the 4SRS method. The final remarks are presented in section 3.8.

3.2 From Business Process Models to Software Models

This section refers to approaches that, in one way or another, relate the business processes modeling and software modeling.

- Korherr and List (Korherr and List, 2006), align software with business processes through the relationship between EPCs and UML 2.

The authors establish a link between the business process activities (in EPCs) and the UML use cases and component diagrams. The use case diagrams are used to identify the software requirements, whereas the component diagram represents the structure of the computer system modules. Use cases diagram represent the software in an abstract way, while the components diagram represent the software physical aspects (Korherr and List, 2006). The authors use the dependencies to relate the use cases actions with the components, since the dependencies enable us to relate behavioral diagrams with elements of UML structural diagrams (Korherr and List, 2006).

- Liang *et al.* propose a method to map automatically the changes made on BPMN to BPEL in order to support and enable real-time BPEL reconfiguration (Liang et al., 2008).

The method builds a relationship between the business process key points and the BPEL process. Thus, it helps users to adjust the business process with simple operations and automatically map these changes to BPEL (Liang et al., 2008).

BPMN provides a set of graphic symbols which make the construction of a business process a relatively easy task, however, users need to do a large number of configurations to map these changes to the BPEL process (Liang et al., 2008). BPEL is a language for the specification of executable business processes, so it is not easily understood by stakeholders from business area (Liang et al., 2008).

The authors also present an algorithm designed to ensure the consistency of the changes made to the BPEL standard.

- The requirements engineering is a key component in the software development process (Castro et al., 2000). However, in Castro *et al.* opinion, the UML is not well prepared to model the initial requirements (*early phase*). The UML is best suited to capture the final requirements (*late phase*) of a system.

The initial requirements are identified at an early stage and are typically informal, non-functional and organization oriented. The final requirements

are identified at a later stage and are usually complete, consistent and targeted for automatic verification of requirements. Thus, the authors propose a set of general guide lines to transform the initial requirements into final requirements.

For the initial requirements specification, the authors use the i^* technique and for the final requirements specification they adopted the pUML (precise UML), which is used in the OCL (Object Constraint Language) for specifying constraints such as invariants and pre-conditions. The i^* is a technique that provides a good understanding of the organization relationships in the business domain (Castro et al., 2000). The pUML provides semantics for the UML basic elements, such as relationships, associations and generalizations. The OCL is a textual language that allows to describe in a precise way the constraints of an object-oriented model (Castro et al., 2000).

The transformation process follows six general lines related to: actors, tasks, resources, goals, decomposition tasks and links. Briefly, actors, tasks and resources are transformed into classes, goals are mapped to class attributes. The decomposition tasks, pre and post conditions are expressed in OCL.

- Martinez *et al.* propose an approach for obtaining a information system conceptual scheme based on the organizational model. The organizational model is specified in i^* (Martinez et al., 2003).
- Truscan *et al.* suggest an approach to integrate DFDs and some UML diagrams, namely the use case diagram, object diagram and class diagram (Truscan et al., 2004). Thus, the authors propose, among others, the following transformations between models:
 - Use case diagram and object diagram;
 - DFDs and object diagram;
 - DFDs and class diagram;
 - Object diagram and DFDs.

The transformations considered of most interest are detailed below. For the transformation of use cases into object diagram, the authors opt by the following approach:

- Firstly, each actor in use case diagram is transformed into an instance of the object diagram.
- For each use case three objects are created in the object diagram: interface, data and control.

- In the objects diagram, the associations are created between the interface object and control object from the same use case. The same happens, in the opposite direction, between the interface object and the control object of the same use case.
- Finally, an association between the use case actor and interface object generated by the use case is created.

To obtain the object diagram from DFDs, the authors propose the following approach:

- Each data processing element in DFD gives rise to a class instance in object diagram.
- Associations between objects come from the DFD flow transformation: class instances originating from the process that receives an input stream are complemented with a method *set()* and the corresponding attributes.
- The method *run()* is appended to the active objects
- The method *send()* is added to the objects that send a flow
- Each data repository is transformed into a class/object with methods *read()* or *write()* depending on the direction of the flow that connects to the data store (output/input).

To obtain the class diagram the authors focus on data, namely on data stores and data flow. So, they start by classifying the data flows and data stores according to a type. Each type identified in DFD gives rise to a class of the class diagram. Each data repository gives rise to a separate class as well as each external entity. The methods are identified based on the data flows between processes, external entities and data stores.

- Rungworawut and Senivongse propose a set of guide lines for obtaining the UML class diagram from a business process. To model the business process the authors use the BPMN language (Rungworawut and Senivongse, 2005).

In this approach, the classes identification is based on the identification of key concepts. The strategy to identify the key concepts involves research on the domain knowledge data bases to identify potential classes for the application. In the last phase, the software modelers can select among the potential class which will actually be part of the class diagram.

The same authors propose an enrichment of the approach presented above in (Rungworawut and Senivongse, 2006).

- Rodrigues et al. propose a model transformation approach using QVT (Query View Transformer) and apply it to BPMN to UML activity diagram

transformation and in UML activity diagram to class diagram transformation (Rodríguez et al., 2010). QVT is a transformation language proposed by OMG (OMG, 2011b). In general terms a QVT transformation starts by identifying the source model (or language) meta-model and the destination model (language) meta-model. Each pattern identified in the source model is transformed into one or more target meta-class instances in the target model, by applying a set of rules and constraints. The rules can be specified or written using declarative or imperative syntax. Constraints can be specified using OCL.

To transform BPMN diagrams into an activity diagram the authors identified the BPMN meta-model, the UML Activity Diagram meta-model and a set of seven rules. The same steps were followed to obtain a class diagram from an activity diagram: the activity diagram and the class diagrams meta-models were defined and also a set of five rules to transform elements defined in the activity diagram meta-model (source) into other elements defined in class diagram meta-model (target).

3.3 From Business Process Models to Data Models

The importance of data in business process modeling is increasing motivated by the need of controlling the business process and the improvement of technical BI (Business Intelligence) whose results are used by organizations to support and planning business (Meyer et al., 2011).

Data are not the focus of business process modelers nevertheless, all business process involves information that must be kept in a persistent manner. Thus, some authors try to obtain the data model based on business process models. Some of these approaches are described below.

- Brambilla *et al.* (Brambilla et al., 2008) explore BPMN for the generation of the data design, business logic, communication and representation. The authors separate the different concerns in different model types and interpret the BPMN in order to meet the needs of a Rich Internet Application (RIA). With respect to the data, the authors use BPMN data objects to identify the data involved. To distinguish between persistent and volatile data, the authors have chosen to identify, in the process model itself, persistent data with a ‘P’ and volatile data with a ‘V’. In BPMN most recent version (BPMN2.0) the notion of persistent data can be represented by a data store (OMG, 2011a).
- Magnani and Montesi (Magnani and Montesi, 2009), after identifying the gaps in data modeling using BPMN, proposed an extension to BPMN 1.2

with the aim of improving the representation of data. The extension was named BPDMN (Business Process and Data Modeling Notation). Some of the concepts proposed, namely a way to identify the existence of persistent data were included in BPMN 2.0 (Magnani and Montesi, 2009) with the introduction of the data store, although with a different graphic symbol.

- Wohed *et al.* (Wohed et al., 2006) make an assessment of BPMN capabilities, its strengths and weaknesses, to model a business process and conclude that, in BPMN 1.2, data is only partially represented.
- The approach presented by de la Vara *et al.* extends the BPMN 1.2 with task description improvements. The approach also presents guidelines for the specification of the domain classes diagram (de la Vara et al., 2009). The approach works with the BPMN 1.2 version where the distinction between persistent and non-persistent data was not possible. Meyer *et al.* extend BPMN data objects with annotations to allow data dependency representation and data instance differentiation (Meyer et al., 2013). The presented approach is able to generate SQL queries from BPMN data objects (Meyer et al., 2013).
- Brdjanin *et al.* propose an approach to obtain a database design based on the information existing in a UML activity diagram (Brdjanin et al., 2011). The authors propose a direct mapping of all business objects to the respective classes. Each participant is also mapped to a class. Associations between business objects and business process participants are based on the activities performed on those objects (Brdjanin et al., 2011).
- Arnon Sturm suggests a method for building a data warehouse schema for specifying business processes in order to allow the off-line analysis of business processes' execution. The data warehouse schema is composed of a small set of "snowflakes" (Sturm, 2008). A snowflake is the basic structure of the data warehouse. For each data object affected by the process a snowflake schema is created. The method comprises two steps: the first step is the creation of the data schema of each snowflake, the second populates the database with relevant data. The proposed method can generate schemas for all "snowflakes" of the business process specification. However, the creation of multiple schemes will overwhelm the design of the warehouse, so Arnon Sturm argues that one must carefully select the business processes to consider.

3.4 From Business Process Models to Use Case Models

UML use case models are one of the most popular adopted techniques to capture and describe the functional requirements of a system.

The use case model is used in requirements elicitation and specification as a means to facilitate the dialog with the customer about “what” the system is supposed to do (OMG, 2012). But a use case model can take months or even years to complete (Cockburn, 2001) and usually involve many resources. To solve this problem some authors have developed approaches to generate use case diagrams based on the business process modeling.

The most relevant of the existing approaches that approximate the use case models and the business process modeling are presented next:

- Lübke and Schneider propose an approach to generate a business process model (modeled in BPMN) from UML use case diagrams (Lubke et al., 2008). The authors justify the need for this approach with the increasing number of use cases and with the possibility of losing the execution order of the various use cases (Lubke et al., 2008).

For the generation of the business process in BPMN, from a set of use cases, the authors propose the following approach (Lubke et al., 2008):

- The first step is to generate a plan use case diagram, i.e., all *include* and *extend* use case relationships are replaced by different scenarios of these use cases.
 - Then a BPMN process is generated for each use case.
 - In the next step, all BPMN processes generated are grouped. This is based on the preconditions and on the success guarantees of the respective use cases.
 - Finally, the resulting BPMN process is restructured, i.e. when there are multiple representations of a constructor only one remains.
- Dijkman and Joosten propose an approach for mapping a business process models to use case diagrams (Dijkman and Joosten, 2002b). For the representation of the business process model the authors choose to use the UML activity diagram.
 - Dijkman and Joosten also propose an algorithm for obtaining the use cases diagram from the business process modeled with the UML activity diagram (Dijkman and Joosten, 2002a). To achieve their goals, Dijkman and Joosten start by defining the activity diagram and the use case diagram meta-models. Then, the authors establish a relation between the “role”

from the activity diagram and the “actor” in a use case diagram and a “step” (a sequence of tasks) from the activity diagram originates a “use case” in a use case diagram (Dijkman and Joosten, 2002a). A step is defined as a sequence of tasks that can be grouped into sub-diagrams. Each step gives rise to a use case. The sub-diagrams are used to detail the use cases. It is created an association between the actor and the use case where there is an association between the role representing the actor and the step that is represented in the use case. The mapping between the two associations is represented by an OCL constraint (Dijkman and Joosten, 2002a).

- Rodriguez *et al.* propose a systematic approach for obtaining the use case and class diagrams from the business process modeling using, once again, the UML activity diagram (Rodríguez et al., 2008). This approach does not allow the identification of the relationship between classes in a complete way, neither the relationship between obtained use cases.
- Rodriguez *et al.* also suggest an approach for the generation of use case diagram based on business process modeling in BPMN (Rodríguez et al., 2007). In this approach a BPMN diagram gives rise to a use case diagram as follows (Rodríguez et al., 2007):
 - The main actor is the participant who initiates the process (lane with the *start event*);
 - The remaining participants are secondary actors;
 - When a *pool* is divided into several *lanes*, each *lane* gives rise to an actor who inherits from the actor originated by the *pool*;
 - A BPMN activity gives rise to a use case in the use case diagram.
- Stolfa and Vondrák propose an approach to generate a use case model based on a business process model using the UML activity diagram (Štolfa and Vondrák, 2008). The approach allows to identify actors, use cases and the relationships (include and extend) between use cases. A swim-lane belonging to an activity diagram is used to represent who is responsible for executing activities. Thus, a swimlane is represented as an actor in use case diagram. Activities are grouped in use cases, allowing to structure use cases by relationships *extends*, *include* and *generalization*.
- Paradkar and Sinha proposed an approach to derive business process models from business use case models (Paradkar and Sinha, 2015). The approach transforms textual use cases descriptions (using Natural Language) into BPMN models. In the presented approach one use case model is mapped to one or more BPMN models. The authors state that both, business process models and use case models, are useful and coexist (Paradkar and Sinha,

2015). The approach helps to maintain consistency between the two forms. This is especially important when the source model is updated (Paradkar and Sinha, 2015). Besides, the generated BPMN models can be used as input to the BPEL “execution engine”.

- Ditze and Henninger propose an approach that combines UML 2.0 with BPMN 2.0 (Ditze and Henninger, 2010). In this approach the BPMN is used to detail, or describe, the use cases from a use case model.

The approach starts by identifying the actors involved in the various business processes. The actors are identified here as those to whom the organization serves. The actors and the different business cases (or use cases) are grouped in a use case diagram. Then each identified use case is detailed using a BPMN model.

In respect to the information involved in the processes, it is modeled using the class diagram. The information exchanged between the actors and the system is modeled in BPMN, through the exchange of messages flowing in collaborations. The data within the processes are modeled on the data objects and data repositories.

All messages, data objects and data repositories must correspond to individual elements in the information model and are represented as a class. Relationships are identified by the data associations in BPMN. In the class diagram relationships are modeled as dependencies.

3.5 From Use Case Models to other Software Models

Model transformation is one of the basic principles of Model Driven Architecture (MDA) (Yue et al., 2011). Several authors propose approaches to derive software models (data models and class models) from requirements representations (use case models).

The use case model describes the functionality of the system while the class diagram describes the system architecture (Fernandes et al., 2000). The class diagram is extremely useful for the static characteristics specification of the software being produced.

Next we describe some approaches that intend to obtain the class diagram based on the use case model:

- In (Christiansen et al., 2007) the authors propose a method to obtain the class diagram directly from the textual description of use cases. To do that, the authors use grammatical analysis of natural language used in the requirements description. Briefly, for a given sentence, the subject of

the sentence is represented by a class, the verb gives rise to a method of the class that represents the subject. The classes attributes are obtained through sentences with verbs like “have”. The relationship between classes is achieved by identifying verbs like “be”.

The authors point to the fact that the syntax of the language used in the use cases textual description shall avoid ambiguity, for example adverbs or adjectives should not be used.

- Santos and Machado propose an approach for obtaining a class diagram based on a use case model by extending the 4SRS method (Santos and Machado, 2010). This approach is detailed in section 3.7.
- Ying and Liang propose an approach for obtaining a class diagram based on a use case model (Ying and Liang, 2003).

The authors begin by identifying candidate classes through the identification of nouns in sentences that describe the use cases. Then select the most significant classes among the candidates. In a later stage, the authors verify (or validate) if the nouns found, on the use case description, fit in the traditional class categories such as roles, concepts, events, etc..

The proposed approach can be summarized in the following steps:

1. Determine the use case main purpose.
 2. Identify entities with the same purpose of the use case.
 3. Specify entities as classes and their properties as attributes of the class.
 4. Specify the associations between classes based on the use cases relationships.
 5. Identify collaboration between use cases and entities.
 6. Specify collaborations as class methods.
- B. Roussev proposes a process for generating formal object-oriented specifications in OCL and class diagrams from the use case model (Roussev, 2003). For this, the author uses a series of well-defined models and transformations and presents an algorithm to convert the use case to a set of OCL expressions. The OCL specification provides a simple language for writing constraints to model elements (Pilone and Pitman, 2005; OMG, 2010b).

The process begins by converting each use case, for each actor, in a state transition machine. To create the class diagram, each actor gives rise to a class, as well as each state through which the process passes. The associations are derived from the relationship between the actors performing the state transition and the states.

The generated OCL constraints allow validating the obtained model. In the obtained model the attributes are not identified neither the classes methods.

A use case model is constituted by actors, use cases and corresponding descriptions. Most of the information is in use case descriptions. Following this idea, some authors proposed approaches to derive software models based on the use cases descriptions dealing with NL. Some of those approaches are presented next:

- Samarasinghe and Somé propose an approach to create a Domain model from a Use Case model (Samarasinghe and Somé, 2005). To describe use cases the authors use a controlled NL to which they propose a grammar. The authors state that automatic processing of use cases described in full NL is not possible so, they propose a restricted form of natural language grammar for use case descriptions (Samarasinghe and Somé, 2005).
- Dražan and Mencl proposed a method for process based in a schema previously defined (Dražan and Mencl, 2007). The proposed method allows dealing with complex structured sentences which may be used to verify the requirements and to derive the initial design of the system. Working with structured sentences in use case descriptions allows automating the textual processing.
- In (Ilieva and Ormandjieva, 2006) the authors propose the generation of the domain model and UML activity diagram from requirements specification. The authors consider unlimited NL, but they rewrite it in a different format using a tabular presentation of the text. They start with a syntactical analysis of the text, and then build a tabular presentation and a semantic network. The construction of models are conducted in four stages: syntax categorization, tabular modeling of the text, semantic processing of the text, and interpretation of the text for diagrammatic modeling (Ilieva and Ormandjieva, 2006).
- Yue *et al.* (Yue et al., 2009) to avoid NL ambiguity, propose a set of restriction rules and a new template to describe use cases. The authors aim is to facilitate the textual analysis, allowing the automatic extraction of the UML class model. The restriction rules and the template are to be used during the requirements elicitation phase.

A restricted and controlled NL reduces ambiguity, redundancy, and complexity of the documents, allowing to automatize the translation into other languages (Bera and Evermann, 2014).

3.6 Use Case Models Decomposition

A use case model may be created with a high abstraction level or with a low abstraction level. Low abstraction level use cases can be much more useful, in software development, than high abstraction level use cases. Some authors propose approaches to decompose the UML use cases model, which are described next:

- Collins-Cope considers three different categories of use cases: requirement, service and interface (Collins-Cope, 1999):
 - The *requirement* use cases, that are used to document the business processes, serving as a guide to the system development.
 - The *service* use cases, which provide a description of the functionality the system will offer. That includes the objective of the service, the list of input and output parameter, pre and post-conditions among other details.
 - The *interface* use cases, that provide a description of the interface presented to the system actors.

Collins-Cope approach uses very detailed use cases and, at the end, these use cases are targeted to be implemented in a three layers architecture product. Nevertheless, the author does not explain how use cases can be managed when this approach is applied in real working scenarios.

- Regnell *et al.* (Regnell et al., 1996) explore the use case model benefits of a graphical representation that have support for descriptions at different levels of abstraction. They propose an approach to decompose use case models that divides the use cases in three abstraction levels:
 - At the *environment level*, each use case is described and associated with related actors. In this level, the use cases can be organized into packages.
 - At the *structured level*, each use case is described as a graph of episodes. One episode is a coherent part of a use case. Each episode is composed by events.
 - At the *event level*, each episode is detailed and described as a flow of events, represented in a sequence diagram.

The authors propose to organize the use cases into packages, at the environment level, to facilitate the use case management. But, in our opinion, at this software development process stage it is too soon to identify the correct packages, because the identified partitions do not obey to any conceptual principle (Berenbach, 2004).

- Glinz states that UML cannot model a rich system context (Glinz, 2000). Glinz identifies several problems and deficiencies of UML, most of them related to use case models and system decomposition. In what concerns to the relationships between use cases, Glinz stands that the UML specification is inconsistent and contradictory (Glinz, 2000). To overcome this deficiency he proposes four new relationships between use cases: sequence, alternative, iteration and concurrency. To overcome the problem concerning use case decomposition, Glinz proposes to decompose the system into several subsystems, and model each subsystem using use case models. However, Glinz concludes that it will originate a very complex structure and that the consistency between use cases cannot be ensured.
- Approaches like (Quartel et al., 1995; Darimont and van Lamsweerde, 1996) were proposed to refine use cases using formal methods. The approach proposed by Quartel *et al.* aims to refine use cases by replacing abstract actions for concrete activities (Quartel et al., 1995). The replacement is based on a list of rules. The approach proposed by Darimont and van Lamsweerde aims to refine by reusing generic refinement patterns (Darimont and van Lamsweerde, 1996). To Greenfield and Short, refinement is a transformation that maps models based on a more abstract modeling language to models based on a more concrete one (Greenfield and Short, 2004).
- Ayman Issa identified several problems in use case modeling. Problems like granularity, inconsistency, and ambiguity are very common because use case modeling methods do not specify the level of detail that should be used in use cases, neither the scope that each use case should cover (Issa, 2007). Ayman Issa proposed an approach to refactor use cases, i.e, to synthesize and refine use case models. The approach is divided in two different types of use case refactoring: behavioral refactoring and structural refactoring. To do so, Ayman Issa itemizes a list of refactoring activities to each one of the refactoring types (Issa, 2007). The identified activities aim to guide the authors to create better use case models. Nevertheless Ayman Issa does not explain how the use cases are refined. The paper focus its attention on the bottom-up approaches, i.e., approaches that start with concrete examples of usage scenarios and generalize them to create the use cases.
- Pons and Kutsche (Pons and Kutsche, 2004) studied the refinement as a way to enable traceability. The refinement is established between class models and between use case models. To refine use cases, Pons and Kutsche use the *include* relationship between use cases.
- Hausmann *et al.* (Hausmann et al., 2002) agree that the decomposition of a complex problem into smaller problems is a very well accepted task, but the re-integration and re-merging of the resulting models is a problematic

and challenging task, especially because the consistency between the models is difficult to maintain. This is particularly important in requirement specification models because the late detection of requirement errors in the development process could cause very expensive re-iterations through all phases (Hausmann et al., 2002).

- Azevedo *et al* propose an extension to the UML use case meta-model for representing a refinement of use cases (Azevedo et al., 2010), but the refine relationship presented by Azevedo *et al* is between two use cases. Azevedo *et al* highlight the differences between *include* and *refine* relationships (Azevedo et al., 2010). The *refine* relationship implies lowering the abstraction level, the *include* relationship does not imply increasing the detail level.
- Cockburn (Cockburn, 2001) does not present an approach to decompose use cases but he distinguishes different use case abstraction levels by assigning different colors to the use cases. The author discerns black and white use cases. The first being low detail level use cases, and the latter being very detailed use cases with a clear intention. Cockburn claims that “non-white” use cases are not trustable as functional requirements for the system being built (Cockburn, 2001).

From another point of view, Cockburn categorizes use cases as business use cases and system use cases (Cockburn, 2001). Cockburn sees business use cases as black use cases and system uses cases as white use cases and advises the use case writers to start by the business use cases and “unfolding” them continuously until they become system use cases (Cockburn, 2001). The approach presented in chapter 6 is aligned with Cockburn’s points of view, helping use case writers in transforming high level (black) use cases into low-level (white) use cases, and helping them in keeping track of the transformations (unfolding) of the use cases.

3.7 The 4SRS method

The 4SRS (Four Step Rule Set) is a method organized in four steps to transform use cases into architectural elements (Machado et al., 2005). Use cases do not have a direct mapping to architectural elements, since a use case can give rise to various architectural elements and several use cases can give rise to the same architectural element (Fernandes et al., 2000). The method consists in the following set of four steps (Machado et al., 2005; Machado et al., 2006; Ferreira et al., 2012):

- Step 1: *Architectural elements identification or component creation* - each use case is transformed into three architectural elements: one interface, one data and one control element. Each element is labeled with the name of the use case followed by the appropriate type (i-interface, d-data and c-control);

- Step 2: *Architectural elements elimination* - based on the textual description of each use case, it is necessary to decide which of the three elements created in step 1, must be maintained; This step is divided into seven micro steps:
 - Step 2i: *Use cases classification* - In this micro step each use case is classified. With the three types of architectural elements identified in the previous step, it is possible to form eight different combinations (0, i, c, d, ic, di, cd and dci). This classification aims to facilitate the transformation of each use case in architectural elements as it provides clues about which categories of elements to use and how they relate.
 - Step 2ii: *Local elimination* - the purpose of this micro step is to check if each architectural element created in step 1 makes sense for the problem domain. Those that do not make sense should be eliminated.
 - Step 2iii: *Architectural element naming* - Each architectural element created, and that has not been eliminated in the micro step 2ii, should receive a name befitting its original use case as well as the role that it has in the system.
 - Step 2iv: *Architectural element description* - Each architectural element that received a name in the micro step 2iii should be described according to the corresponding system requirements, in order to be included in the logic model (depicted by objects diagram). The description must be based on the original description of the use case.
 - Step 2v: *Architectural element representation* - This micro step, through an analysis of each element, ensures the semantic consistency of logic model, eliminates redundancy and enables the discovery of anomalies in use case models, namely missing requirements.
 - Step 2vi: *global elimination* - In this step all micro architectural elements that are represented by others architectural elements are eliminated, since the requirements that correspond to these architectural elements no longer belong to them.
 - Step 2vii: *Architectural elements renaming* - This micro step aims to rename all the remaining architectural elements. The new name to assign must be according the system requirements related to the architectural element.
- Step 3: *Architectural elements aggregation and packaging* - for architectural elements that remain after the elimination, and those in which it is possible and exist advantages in its unification, are aggregated;
- Step 4: *Architectural elements association* - associations must link the elements resulting from the aggregation based on use cases textual descriptions.

The 4SRS generates the logic architectural model based on user requirements represented through a use case model (Machado et al., 2006). The method applies successive transformations to software architecture in order to meet user requirements.

In (Santos and Machado, 2010), the authors propose an extension to the 4SRS method in order to obtain the class diagram based on the use case model and on the logical architecture that results from the application of the 4SRS method. The proposed extension adds two main steps to 4SRS:

- Step 5: *class creation* - a class is created for each object (or architectural element) on the object diagram.
- Step 6: *class characterization* - in this phase the attributes and methods belonging to each class are identified. This identification is accomplished by analyzing textual description of the requirements made in natural language. This step is divided in two micro-steps:
 - Step 6i: *methods creation* - This step identifies the methods that belong to each class. The methods correspond to verbs presented in use case textual description.
 - Step 6ii: *attributes creation* - This step identifies the attributes that should belong to each class. The attributes correspond to the names existing in the refined use case textual descriptions.

3.8 Final Remarks

As we saw earlier, there are several approaches that try to obtain one model from other models. Other approaches use models to complement other models, as is the case of (Ditze and Henninger, 2010; Lubke et al., 2008) where the BPMN language is used to specify each use case in the use case diagram. Other approaches attempt to automate conversion processes, such as (Liang et al., 2008) approach whose goal is to propagate changes made in BPMN to BPEL.

The approaches that relate, or convert, a model to another are summarized in Table 3.1, presenting the authors, the source model and the derived model(s).

Some of the studies cited above, as is the case of (Ying and Liang, 2003; Roussev, 2003; Christiansen et al., 2007), propose approaches for obtaining the objects diagram and/or classes from use case models.

The connection between use cases and business processes is studied in several ways. Some authors propose approaches to get use cases from business processes models, as is the case of (Rodríguez et al., 2007; Dijkman and Joosten, 2002a). Others try to obtain a business process model from a use case model, as is the case of (Lubke et al., 2008). Others use the use case diagram to describe business processes (Nawrocki et al., 2006).

Approach	Source		Destination
Korherr and List (Korherr and List, 2006)	EPCs	▣▣▣▣▶	UC model Component diagram
Truscan <i>et al.</i> (Truscan et al., 2004)	DFDs	▣▣▣▣▶	Class diagram Object diagram
Rungworawut and Senivongse (Rungworawut and Senivongse, 2005)	BPMN 1.2	▣▣▣▣▶	Class diagram
Rodriguez <i>et al.</i> (Rodríguez et al., 2007)	BPMN 1.2	▣▣▣▣▶	UC model Class diagram
Rodriguez <i>et al.</i> (Rodríguez et al., 2008)	AD (UML)	▣▣▣▣▶	UC model Class diagram
Rodriguez <i>et al.</i> (Rodríguez et al., 2010)	AD (UML)	▣▣▣▣▶	Class diagram
Dijkman and Joosten (Dijkman and Joosten, 2002b)	AD (UML)	▣▣▣▣▶	UC model
Stolfa and Vondrák (Štolfa and Vondrák, 2008)	AD (UML)	▣▣▣▣▶	UC model
Jan Dietz (Dietz, 2003)	DEMO	▣▣▣▣▶	UC model
Christiansen <i>et al.</i> (Christiansen et al., 2007)	UC model	▣▣▣▣▶	Class diagram
4SRS (Santos and Machado, 2010)	UC model	▣▣▣▣▶	Component diagram Class diagram
Samarasinghe and Somé (Samarasinghe and Somé, 2005)	UC model	▣▣▣▣▶	Domain model
Boris Roussev (Roussev, 2003)	UC model	▣▣▣▣▶	Class diagram with OCL constraints
Brdjanin <i>et al.</i> (Brdjanin et al., 2011)	AD (UML)	▣▣▣▣▶	Data model
de la Vara <i>et al.</i> (de la Vara and Sánchez, 2009)	BPMN 1.2	▣▣▣▣▶	Domain model
Paradkar and Sinha (Paradkar and Sinha, 2015)	UC model	▣▣▣▣▶	BPMN
Lübke and Schneider (Lubke et al., 2008)	UC model	▣▣▣▣▶	BPMN

Table 3.1: Summary of approaches that make the relationship between models

Approach	Source	Use cases	Actors	Actors hierarchy	Include Extend	Descriptions
Dijkman and Joosten (Dijkman and Joosten, 2002b)	AD	+	+	-	+	-
Dijkman and Joosten (Dijkman and Joosten, 2002a)	AD	+	+	-	+	-
Rodriguez <i>et al.</i> (Rodríguez et al., 2008)	AD	+	+	-	+	-
Rodriguez <i>et al.</i> (Rodríguez et al., 2007)	BPMN	+	+	+	+	-
Stolfa and Vondrák (Štolfa and Vondrák, 2008)	AD	+	+	-	+	-

Table 3.2: Summary of approaches that obtain the UML use cases model

Approaches that aim to get a use case models based on business process models are summarized in Table 3.2, where a '+' indicates that the approach supports the corresponding use case model element, and a '-' means the opposite. Analyzing the Table 3.2, it can be concluded that none of the approaches presents a proposal for obtaining the use cases description. Nevertheless, the use case descriptions are one the most important part of a use case model. Besides that, none of the cited approaches aggregates a set of business process models in one use case model.

Although some of the presented approaches focus data, none of them is able to derive a complete data model based on business process models. The same conclusion was drawn by Brdjanin and Maric in the survey presented in (Brdjanin and Maric, 2013), which points that none of the existing approaches generates a complete data model based on business process models (Brdjanin and Maric, 2013), including the first approach presented next in this research work, in section 4.2.

Chapter 4

Deriving a Data Model from Business Process Models

Business process modeling and management approaches are increasingly used and disclosed between organizations as a means of optimizing and streamlining the business activities. A business process model identifies the activities, resources and data involved in the creation of a product or service, integrating useful information that can be used to create a data model for the supporting software system. A data model is one of the most important models used in software development.

Among the various existing modeling languages, we stress the BPMN, currently in version 2.0. BPMN is a widespread OMG standard that enables business process modeling, but does not facilitate the modeling of the information infrastructure involved in the process. However, interest in the data and its preservation has increased in BPMN's most recent version.

This chapter first presents an approach to get the data model based on one BPMN model. This approach is then extended and improved to generate a data model, based on a set of interrelated business processes, modeled in the BPMN language. This extension is needed mostly because an organization usually deals with several business processes and therefore a software product does not usually support only one business process, but rather a set of business processes.

This extended approach allows getting a complete data model aggregating all the information about persistent data that can be extracted from the set of business process models, serving as a basis for the development of the software that will support the business.

4.1 Introduction

Markets' globalization and the constant increase of competition between companies demand constant changes in organizations in order to adapt themselves to new circumstances and to implement new strategies. Organizations need to have a clear notion of their internal processes in order to increase their efficiency and

the quality of their products or services, increasing the benefits for their stakeholders. For this reason, many organizations adopt a BPM approach (van der Aa et al., 2015; Redlich et al., 2014; Kalenkova et al., 2014; Alter, 2015).

Among the various existing modeling languages, we opted for the Business Process Model and Notation (BPMN), currently in version 2.0 (OMG, 2011a), because it is a widespread OMG standard that is very well accepted and actually used in companies and organizations (Recker, 2008; Aagesen and Krogstie, 2015; Alter, 2015). Besides, it is a complete language that allows creating detailed business process models (OMG, 2011a).

Usually, a business process model is created with a level of abstraction higher than the one needed in software models (Cockburn, 2001). In BPMN's most recent version, the number of graphical element has increased, including now the data store element representing persistent data (OMG, 2011a). This is an important update if one intends to use BPMN models as a basis for the development of the software product that supports the business processes. The approaches presented in this chapter benefit from detailed business process models, as highly detailed business process yields to more complete data models.

During a business process execution, resources and/or data are used and produced. In fact, the information about the data that flows through the process is very important to the software development. The data received, created or used during a process execution can be represented by *message flows* or *data associations* as shown in Table 4.1.

A message flow connects two pools, representing the message exchange between the two participants (OMG, 2011a). Data associations connect activities and data objects or data stores as represented in Table 4.1. A participant represents a role played in the process by a person, an organization's department or something involved in the process. A message represents the content of a communication between two participants.

Data that flows through a process are represented by *data objects*. Persistent data are represented by *data stores*. Persistent data are the ones that remain beyond the process life cycle, or after the process execution ends (OMG, 2011a). Data objects and data stores are exclusively used in private process diagrams (OMG, 2011a). That's the main reason why the first approach presented in this chapter is based on private business processes.

In software development different models are usually used to represent different perspectives. The data model is one of the most important models for designing software applications, representing and organizing data, how it is stored and accessed, and the relationships among different entities.

An entity is something identifiable, or a concept in the real world that is important to the modeling purpose (Weske, 2012). The information, or the properties, about an entity are expressed through a set of attributes (Weske, 2012). A relationship between two entities is represented through an association between those entities (Chen, 1976).



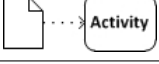
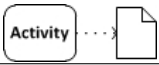
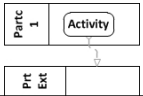
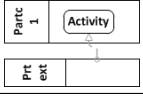
Data	Graphical representation	Meaning
Data Store		The activity reads information from the data store
Data Store		The activity writes information in the data store
Data Object		The activity receives a data object
Data Object		The activity sends a data object
Message		The participant (activity) sends a message to an external participant
Message		The participant (activity) receives a message from an external participant

Table 4.1: The Data handling

A relationship between two entities can be classified in two aspects, Cardinality and Optionality. Both terms are used to denote the number of attributes in a relation. Cardinality represents the maximum number of instances (one or many) of an entity in relation to another entity. Relationship optionality represents the minimum number of elements that exist on that side of the relationship. It may be 1 (the relation is mandatory) or 0 (the relation is not mandatory).

In what concerns to cardinality, three types of relationships can be identified (Chen, 1976): mappings $(1 : n)$, $(m : n)$ and $(1 : 1)$. Focusing in one side of a relationship type, and considering the optionality and cardinality together, we may have: 0 or 1 (represented as $+○-$), 1 ($+\#-$), 0 to many ($-○\infty$) and 1 to many ($-+\infty$).

From a data point of view, a business process model has lots of information that can be used to create a data model. But can a data model be created based on the information we have in a set of interrelated business process models?

This chapter presents approaches for deriving a data model from business process models. First we worked with only one business process model. After that, the approach is extended and generalized to work with a set of interrelated business process models.

The approach to obtain the data model based on a single BPMN private business process model was presented and published in (Cruz et al., 2012). Some

limitations were identified in this approach, namely the impossibility of identifying some of the relationships between the entities represented in the data model, the optionality of some association ends is not addressed, and the most significant limitation is the fact that this approach deals with only one business process model. However, a software application usually supports more than one single business process. Consequently, the approach presented in (Cruz et al., 2012) could not integrate in one data model all the data involved in all business processes that must be supported by the software under development.

The approach was extended and completed allowing the aggregation of the existing information spanning several related business process models into one data model. This extended approach allows identifying the entities involved, the corresponding attributes and the relations, including cardinality and optionality, between the entities.

The approach to obtain the data model based on a set of interrelated business process models was presented and published in (Cruz et al., 2015b). In this improved approach we are able to extract the existing information about data (focusing our attention in persistent data) from a set of interrelated business process models and create a data model that can be used as a basis for developing a supporting software system.

The remainder of this chapter is structured as follows. Next section describes the approach to generate a data model from one business process model. The application of this approach is illustrated through two demonstration cases in section 4.3. Section 4.4 extends the approach for data model creation based on a set of business process models. The application of the extended approach is illustrated through a demonstration case in section 4.5. Finally, section 4.6 analyzes the generated data model, and section 4.7 draws some conclusions.

4.2 Deriving a Data Model from one Business Process Model

Although previous work about data modeling within BPMN already exists (see section 3.3), some previous work is related with BPMN versions prior to 2.0 and, to our knowledge, none of them addresses the attainment of a data model that operationally supports a set of business processes. In some of these studies some flaws have been identified, by the authors, especially in distinguishing persistent from non-persistent data.

In (Borger, 2011), Borger claims that a notion of state is missing in BPMN, and consequently the specification of data dependence conditions is poorly supported. To overcome this fact, we opted by involving BPMN participants in the process. A BPMN participant is always related with all activities where he/she participates and, consequently, with the data stores manipulated by these activ-

ities.

This first approach is based on the following considerations:

- The information about the participants in the process is relevant to the process, especially for its control, so all participants involved in data exchanges shall be represented in the data model.
- The pool representing the organization (company, department, etc.) that is designing the process shall disclose all internal roles involved. The other pools, as they represent entities outside the organization, do not have that need. It is only needed to show the input and output flow of information, e.g. through messages.
- Since a data store represents the persistent data, all data stores involved in the process shall be represented in the data model.
- If one wants to focus on different states in the same data store, the data store name shall be given by $\langle DataStoreName \rangle [\langle DataStoreReferenceState \rangle]$, similarly to what happens with the data object.
- If there is involvement of data within a sub-process, the data flow shall be viewed with the subprocess extended, i.e., within the subprocess.
- Data objects may represent electronic data as well as physical data. However, in both cases data must be stored. For example, if a data object represents the arrival of a shipment, the data about the shipment must be stored.

Our approach is organized into a set of three groups of rules. Each group is devoted to a particular goal: the first group (R1) identifies the data model entities, the second group (R2) identifies the relationships between entities, and in the third group (R3) the entities' attributes are identified.

The first group of rules is explained below:

- R1.1: Each data store, identified by a name, must be represented by an entity in the data model. The entity name is the name of the data store.
- R1.2: When two data stores have the same name, it is considered that they represent the same data store, so it will be represented by the same entity in the data model.
- R1.3: A role played by a participant (represented by a Lane or a Pool) must be represented by an entity in the data model:
 - If the pool is divided into several lanes, each lane will be represented by a data model entity. The name of each entity will be the name of the corresponding lane.

- If the pool is not divided, the pool will result in an entity. The entity name will be the Pool name.
- R1.4: If a participant has the same name as a data store it will be represented by the same entity in the data model.

The second group of rules is explained below:

- R2.1: When a participant is responsible for an activity that manipulates a data store, the entity that represents the participant must be related with the entity that represents the data store. Each participant can perform the same activity several times, so the relationship between the entity that represents the participant and the entity that represents the data store, by default, will be $(1 : n)$.
- R2.2: If, in R2.1, the activity that handles the data store is a cyclical activity, or “Multi-instantiable”, i.e., it may be repeated several times within the process instance, then the relationship between the entity that represents the participant and the entity that represents the data store is $(m : n)$.
- R2.3: If the activity that handles the data store sends or receives information to/from another participant, this means that the participant provides or uses information from that data store. Therefore, there must be an “indirect” relationship between the entity that represents this participant and the entity that represents the data store. By default, the relationship type will be $(1 : n)$.
- R2.4: If, in R2.3, the activity fulfills the conditions presented in R2.2, the relationship type will be $(m : n)$.

Two rules have been created to identify the entities attributes. One to identify the attributes of the entities that represent the participants and the other to identify the attributes of the entities that represent the data stores:

- R3.1: A data store is an “Item-Aware element”, so, as mentioned before, the data structure definition of these elements could be specified as a XML file. The definition of the structure will be used to identify each item that belongs to the data store. Each item identifies an attribute of the entity that represents the data store.
- R3.2: In Swimlanes only the name is identified. Consequently, the attributes of an entity that represents a role played by a participant are static and are described next:
 - ID - it represents the participant identification (code number). It could be the number of the employee, the code of business partner, etc.

- Name - it represents the participant name, for example the name of the employee.

In the presented approach, the data stores and the roles played by participants give origin to entities in the data model. The relationship between the identified entities is deduced from the information exchanged between participants and the activities that manipulate the data store in two ways: directly by the participant that performs the activity and indirectly by the participant that sends or receives information to/from the activity that operates the data store.

From the business modeling point of view, it can be argued that the business process perspective is affected because the model complexity is increased and that the description of the business process essential features can become vaguer due to the introduction of this extra data modeling perspective. A modeler either focuses on the business process description, or he/she focuses on the data model perspective, but not in both of them at the same time. So, this approach should be applied as a further step to modeling the business process and the resulting model should be kept as a branch.

4.3 Demonstration Cases with one Business Process Model

In this section, two examples demonstrating the application of the presented approach are described.

Doctor’s Office Demonstration case

Figure 4.1, represents a process of appointment scheduling and attendance at a doctor’s office.

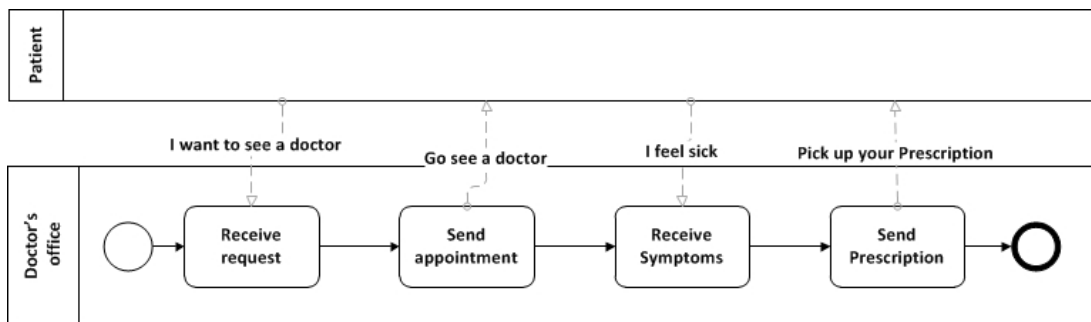


Figure 4.1: A BPMN process example (adapted from (OMG, 2011a))

The same diagram can be redrawn in order to consider the related data. Some information involved in the process must be stored persistently, including scheduled appointments, the symptoms and the made prescriptions. It is needed to

store the appointments scheduled in order to avoid conflict with new appointments. For this reason whenever it is required to make a new appointment it is necessary to check the appointments already scheduled. It is also necessary to store the symptoms and prescribed medicines in order to maintain the patient's history.

Within the Pool *Doctor's Office* there are two roles involved, or two types of participants in the process: the receptionist and the doctor. Thus, we can increase the level of detail of the diagram in order to assign activities to each one of the two involved participants. Thus, the diagram could be redrawn as shown in Figure 4.2.

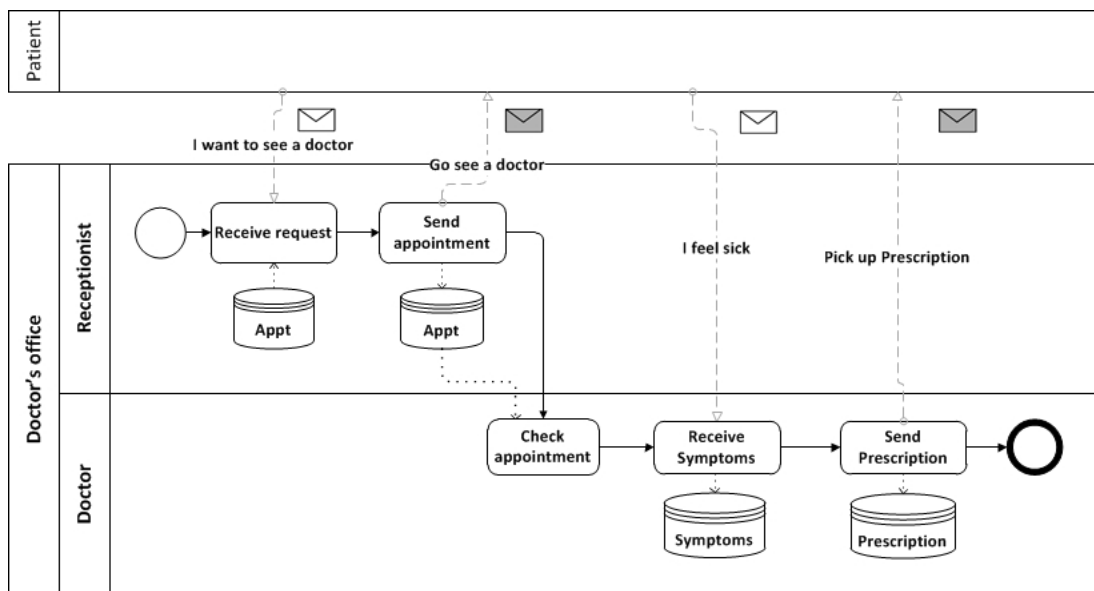


Figure 4.2: Business process diagram focused on data

Based on the diagram shown in Figure 4.2, the following entities can be identified: Receptionist, Patient and Doctor originated from participants (R1.3) and Appt, Symptoms and Prescription originated from data stores (R1.1 and R1.2).

The relationship between the entities is described next:

- The *Receptionist* reads and writes the data store *Appt*, so there is a relationship between the entity Receptionist and the entity Appt. By R2.1, the relationship type is $(1 : n)$. A receptionist can make several appointments. An appointment is made by one receptionist.
- The *Doctor* participant handles *Symptoms*, *Prescription* and *Appt* data stores, so there is a relationship between the entity Doctor and each one of the entities Symptoms, Prescription and Appt.
 - Since a *Doctor* can perform the same activities several times, by R2.1, the relationship between Doctor and Symptoms entities is $(1 : n)$.

- For the same reason, the relationship between Doctor and Prescription entities, is $(1 : n)$ as well as the relationship between Doctor and Appt.
- The *Patient* participant has an “indirect relationship” with the *Appt* and *Symptoms* data stores, since the activities that manipulate these data stores are activated by messages sent by the *Patient*. Once a patient can perform the same activities several times, by R2.3, the relationship between the entity Patient and the entity Appt is $(1 : n)$. For the same reason, the relationship between the entities Patient and Symptoms is $(1 : n)$.
- The *Patient* participant receives a message as a result of the activity that manipulates the data store *Prescription*. So, by R2.3, the relationship between the entity Patient and the entity Prescription is $(1 : n)$.

Figure 4.3 shows the data model resulting from the application of our approach to the diagram shown in Figure 4.2.

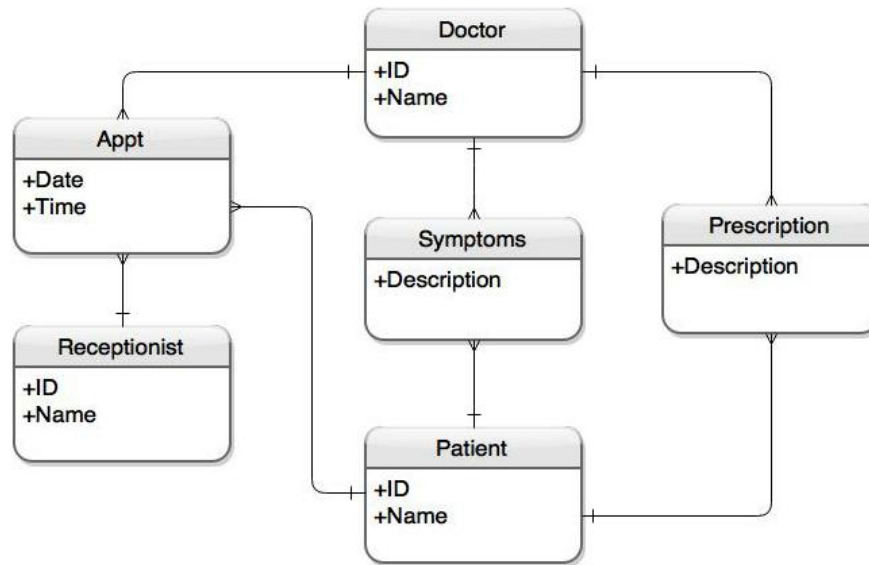


Figure 4.3: Doctor’s Office Data Model

By R3.2, the entities Receptionist, Patient and Doctor will have the same attributes: ID and Name;

By R3.1, the attributes of the entities appt, Symptoms and Prescription are detailed in a XML file.

Nobel Prize Demonstration case

The diagram shown in Figure 4.4, adapted from (OMG, 2010a), represents the Nobel Prize in Medicine Process Diagram.

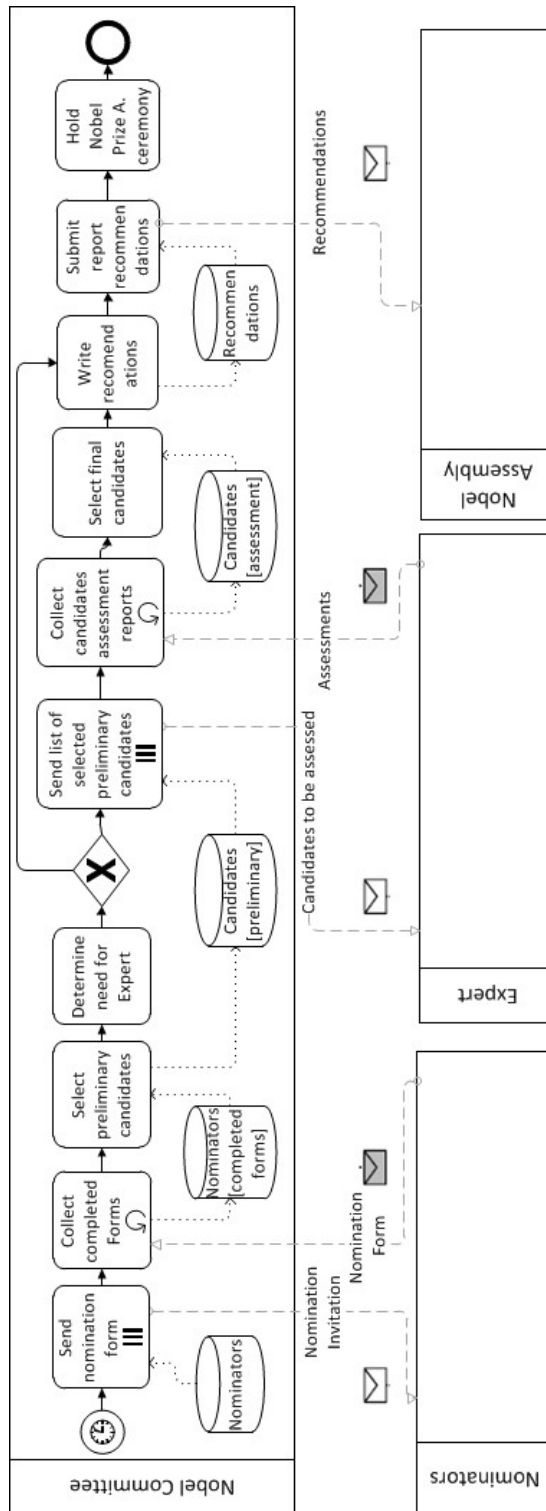


Figure 4.4: The Nobel Prize Process Diagram (adapted from (OMG, 2010a))

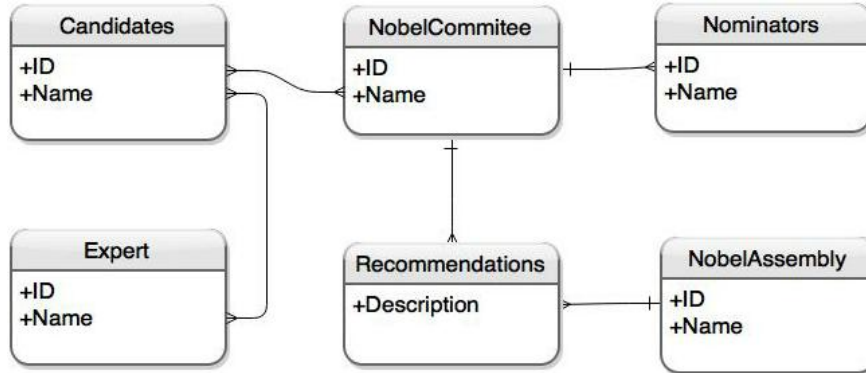


Figure 4.5: Nobel Prize Data Model

As we can see in Figure 4.4, there are four roles, or participants, involved (Nobelcommittee, NobelAssembly, Nominators and Experts) and three different data stores (Nominators, Candidates and Recommendations). By R1.3, each participant will be represented by an entity in the data model. By R1.1, each data store will be represented as an entity in the data model. But, the *Nominators* participant has the same name as the *Nominators* data store, so, by R1.4, both are represented by the same entity (Nominators) in the data model.

Figure 4.5 shows the data model resulting from the application of the approach to the diagram shown in Figure 4.4.

The relationship between the entities is described next:

- The activities that manipulates the *Nominators* data store are cyclic, so by R2.2, the relationship between *Nobelcommittee* and *Nominators* is $(m : n)$. The same happens between the entities *Nobelcommittee* and *Candidates*.
- By R2.1, the relationship between the entities *Nobelcommittee* and *Recommendations* is $(1 : n)$.
- The activities that manipulates the *Candidates* data store are cyclic and exchange information with the *Expert* participant, so by R2.4, the relationship between the entities *Candidates* and *Expert* is $(m : n)$.
- By R2.3, the relationship between the entities *NobelAssembly* and *Recommendations*, is $(1 : n)$.

By R3.1, the attributes of the entities Nominators, Candidates and Recommendations are detailed in a XML format file.

By R3.2, the entities Nobelcommittee, NobelAssembly, Nominators and Experts will have the same attributes: ID and Name.

Since *Nominators* is a participant and a data store, its attributes will be the joint of all.

4.4 Deriving a Data Model from a Set of Inter-related Business Process Models

Typically, in a real situation, a software product does not support only one process, but rather a set of processes. So, in order to generate a data model useful for the development of such software product, it is necessary to assemble all data models resulting from the application of the previous approach to each process being supported.

This section presents an extension to the approach presented previously (section 4.2). This extension, intends to enable the derivation of a data model directly from a set of interrelated business process models by aggregating and merging the information about the data derived from a set of interrelated business processes.

To do that, first it is needed to identify and specify which business processes are to be supported by the software under development, identifying the system scope (Cruz et al., 2015a). Then it is necessary to group, aggregate and merge in one data model all the information about data derived from those business processes.

When we are working with a set of interrelated business process models it is natural to find a participant involved in several business processes. Following the same idea, we can say that a data store can also be involved in several business processes.

Usually, interrelated business processes complement each other, meaning for example that when a business process reads information from a data store, that information can be written during the execution of the same or another related business process.

Assumptions

The approach here proposed assumes that:

- A set of private business processes is considered, and, in this scenario, each identified business process is represented in one (main) pool that can be divided in several lanes. The other participants (pools) involved in the business process are considered as “external participants”.
- When an activity receives information (messages) from an “external participant” and when that information must be kept beyond the process execution, that activity must write the received information into a data store.
- A data object represents data (a document, etc.) that an activity sends or receives. When the information contained in that data object must be kept beyond the process life cycle, the activity must write the information in a data store.

- When a business process reads information from a data store and no other business process writes information in this data store, this can mean that something is wrong (for example a business process is missing) or that a link with another application exists. The same happens when a business process writes information in a data store that is never used (or no other activity reads information from that data store).

Rules to generate the Data Model

To define a persistent data model one needs to identify the domain entities, their attributes, and the relationships between those entities (Weske, 2012). Following this reasoning, the approach is divided in three parts: first present a set of rules to identify the entities, then the entities' attributes are identified, and, finally, we present a set of rules to identify the relationships between the identified entities, including cardinality and optionality. The rules are in accordance with, and extend, the rules already presented in section 4.2. Some of the rules are rewritten and modified to be applied to a set of business processes.

The first set of rules, to identify the entities, is:

- R1: A data store, belonging to one of the selected business process models, is represented by an entity (with the same name) in the data model.
- R2: Data stores with the same name, involved in several business processes, are represented by the same entity (with the same name) in the data model.

When data stores with the same name are involved in more than one business process we assume that they represent the same data store, so they will be represented by the same entity in the generated data model.

- R3: Each participant involved in one of the selected business process models originates an entity in the data model.
- R4: Participants with the same name, involved in more than one business process, are represented by the same entity (with the same name) in the data model.

Usually a participant is involved in several business processes of an organization. When participants with the same name are involved in more than one business process we assume that they represent the same participant, so they will be represented by the same entity in the data model.

- R5: Participants with the same name as data stores are represented by the same entity (with the same name) in the data model.

When we are working with several related business processes usually the information about “external participants” is stored during one of those

processes. Afterwards, when joining a group of related business processes, we can find participants and data stores with the same name. In that case, they will be represented by the same entity in the data model.

The rules to identify the entities' attributes are:

- R6: The attributes of an entity derived from a data store are the integration of all attributes involved in all the data store manipulations during the business processes execution.

As explained previously in section 4.2 each item from a data store is represented as an entity attribute in the data model. A data store can be manipulated by several activities belonging to different (or the same) business processes. Each activity may involve different items giving origin to different attributes of the entity that represents the data store in the data model. To prevent loss of information, all involved items must be represented as entity attributes in the data model. We assume that attributes with the same name represent the same attribute.

- R7: The initial attributes of an entity that represents a participant (involved in one, or several business processes) are *id* and *name*. When a participant is involved in several business processes, the participant's name is the same, so the entity attributes are also the same (*id* and *name*).
- R8: When an entity represents a participant and a data store, the entity will aggregate all the attributes representing all the items involved in data store manipulations and the attributes belonging to the participant's representation (*id* and *name*).

The rules to identify the relationships between the identified entities are explained next.

- R9: When a participant is responsible for an activity that writes information in a data store, the entity that represents the participant must be related with the entity that represents the data store.

A participant can perform a process (and an activity belonging to that process) several times, so the relationship is (1:n) from the entity that represents the participant to the entity that represents the data store.

The relationship is mandatory on the side of the entity that represents the participant because the activity is always executed by someone playing that role. Nevertheless it is not mandatory on the side of the entity that represents the data store because this process may never be executed by a particular participant. But the participant can be involved in other processes.

- R10: When an activity that handles the data store exchanges information with an “external participant” we can distinguish four different cases:
 1. The activity receives information from the participant and writes information in a data store (see example in Figure 4.6). In this case, we assume that the information stored is provided by the participant so, the entity that represents the data store is related with the entity that represents the participant.

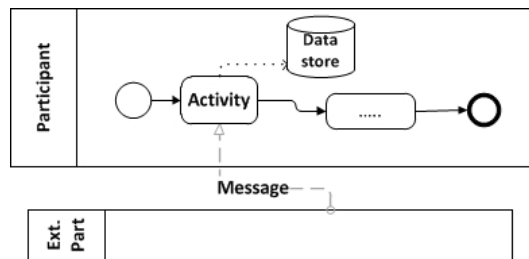


Figure 4.6: Receiving information from a participant

2. The activity writes information in a data store and sends information to the participants. We assume that the same information is stored and sent to the participant (as for example a receipt or a certificate) so, the entity that represents the data store is related with the entity that represents the participant.
3. The activity reads information from a data store and sends the information to an external participant (see Figure 4.7). We assume that the read information is provided to the participant so, the entity that represents the data store is related with the entity that represents the participant.

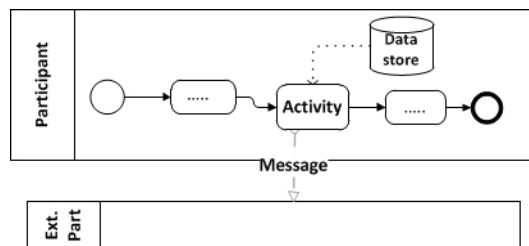


Figure 4.7: Sending information to a participant

4. When an activity reads information from a data store and also receives information from a participant we assume that the activity is only checking the information provided by the participant which is already stored (probably during the execution of another process). In this case, there is no relationship between the identified entities.

In points 1, 2 and 3, the identified relationship type is (1:n) from the entity that represents the participant to the entity that represents the data store (Cruz et al., 2012). Nevertheless, when the activity that sends a message to a participant is a looping activity, the relationship type is (m:n) (Cruz et al., 2012). A Loop Activity is an activity with looping behavior (cyclical or “multi-instantiable”) (OMG, 2011a) meaning that the activity can be executed several times during one process instance. Each Loop Activity instance may have different values to the identified attributes. So, if one loop activity is sending a message to an external participant it may represent the information being sent to several participants.

The relationship is mandatory on the side of the entity that represents the participant because the activity always interacts with someone playing that role. On the side of the entity that represents the data store it is not mandatory, because this process may never be executed by a particular participant.

- R11: When, during a process, an activity writes information in a data store and, in a same process, a previous activity (or the same activity) reads information from another data store (Figure 4.8), and between the activities the process does not receive any other information, it is assumed that the read and the written information are related. As a consequence, the two entities (representing the two data stores) are related.

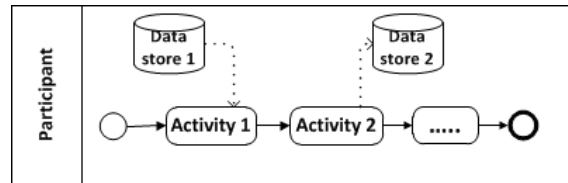


Figure 4.8: Relating two data stores

As a process can be executed several times, the same information can be read several times, so by default the relationship type is (1:n) from the entity that represents the read data store to the entity that represents the written data store. If the activity that writes information is a looping activity the relationship type is (m:n).

By default, the relationship is mandatory on both sides. But, if the execution of the activity that writes the information depends on a condition, for example an exclusive decision gateway (see example in Figure 4.9), then the relationship is not mandatory on the side of the entity representing the written data store.

If the activity that writes information is performed after a merging gateway (not a parallel join) (Figure 4.10), then the relationship is not mandatory on

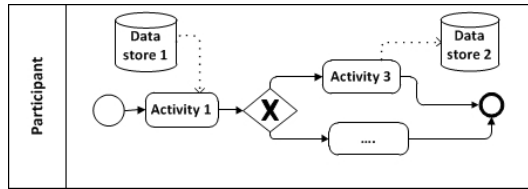


Figure 4.9: Exclusive decision gateway example

the side of the entity representing the read data store because the activity that reads the data store may not be executed.

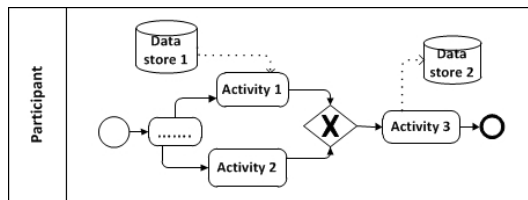


Figure 4.10: Exclusive merging gateway example

- R12: All the relationships derived from the several business processes must be preserved in the data model.

A data store may be manipulated by several activities belonging to distinct (or to the same) business processes, giving origin to different relationships. To prevent the loss of information all the relationships must be represented in the generated data model (for further evaluation).

- R13: If, between two entities, there are different relationship types, the relationship type with higher cardinality prevails.

A data store can be manipulated by different activities. If an activity is a looping activity and another activity is not, it will originate relationship types with different cardinalities. In that case, the relationship with higher cardinality prevails.

- R14: If, between two entities, there are relationships with different mandatory types, the not mandatory type prevails.

4.5 Demonstration Case with a Set of Business Process Models

In this section we use, as a demonstration case, a very well-known example of a School Library System where a group of five related business process models (in

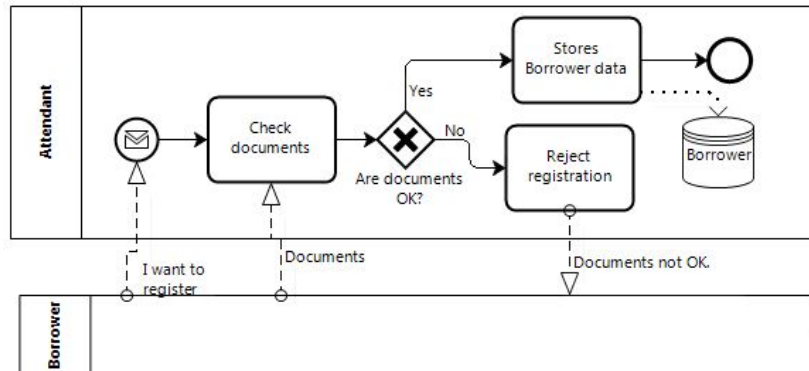


Figure 4.11: Register User business process model

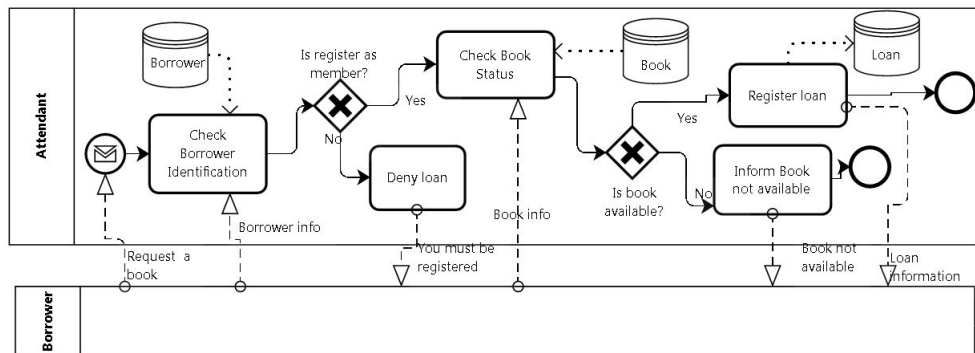


Figure 4.12: Lend a Book business process model

BPMN) have been selected to be presented here. The selected business processes are: *Register User* (Figure 4.11), *Lend a Book* (Figure 4.12), *Reserve a Book* (Figure 4.13), *Renew a Loan* (Figure 4.14) and *Return a Book* (Figure 4.15). The *Return a Book* business process model includes a sub-process, *Penalty treatment* represented in Figure 4.16. In this example, we are going to see that the business processes do not totally complement each other.

The participants involved in all the five selected business processes are the same: *Borrower* and *Attendant*. The two corresponding entities, with the same name, must be represented in the resulting data model.

The *Register User* business process model (Figure 4.11) stores information in the *Borrower* data store, so the *Borrower* must be represented as an entity in the data model. The *Borrower* entity has been identified previously as representing a participant so, by R3, the data store and the participant will be represented by the same entity (*Borrower*). The *Attendant* participant is responsible for executing the activity that writes information in the *Borrower* data store so, by R9, the *Attendant* entity is related with the *Borrower* entity. The relationship type is (1:n) and it is not mandatory on the side of the *Borrower* entity.

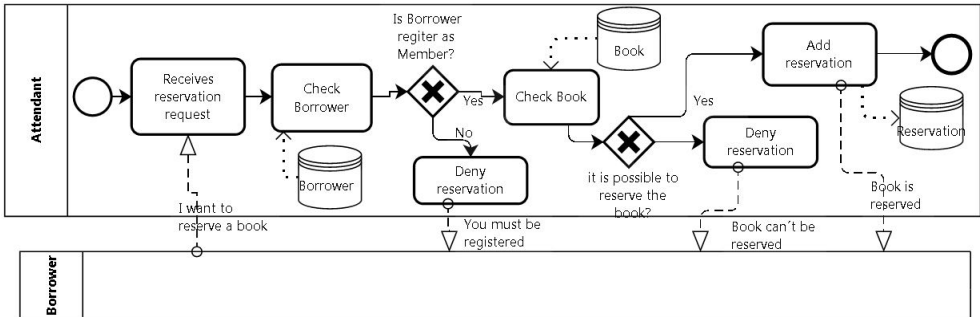


Figure 4.13: Reserve a Book business process model

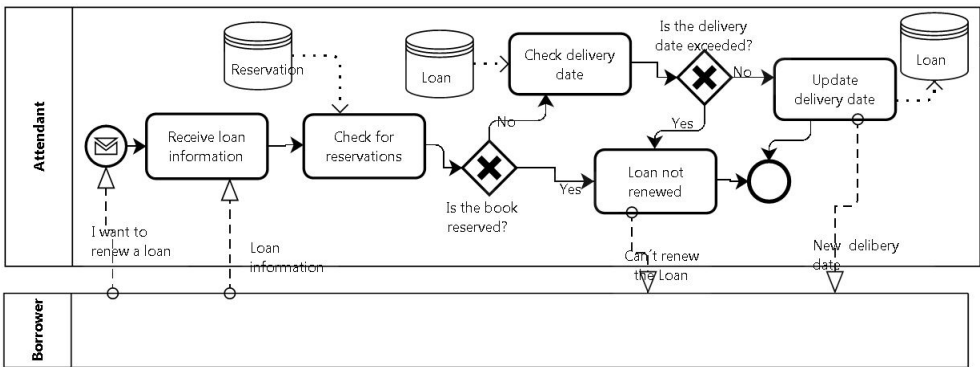


Figure 4.14: Renew a Loan business process model

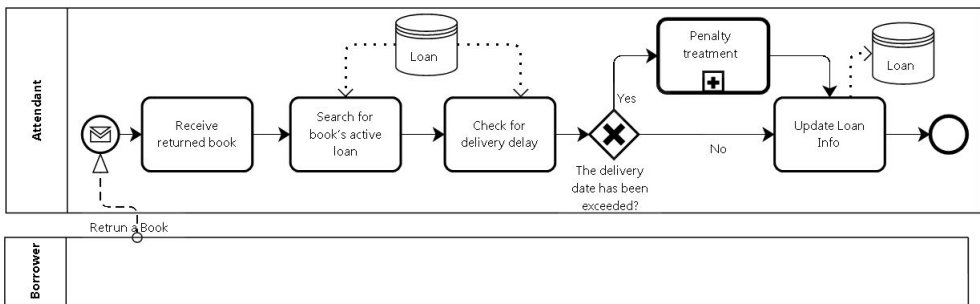


Figure 4.15: Return a Book business process model

In the *Lend a Book* business process model (Figure 4.12) three data stores are involved: *Borrower*, *Book* and *Loan*. *Borrower* has already been identified as an entity. *Book* and *Loan* will originate new entities, with the corresponding name, in the data model. The *Attendant* participant is responsible for writing information in *Loan* data store so, by R9, the *Attendant* entity is related with *Loan* entity. The relationship type is (1:n).

In the same business process model (*Lend a Book*), the *Register Loan* activity sends information to the *Loan* data store and sends a message about it to the *Borrower* participant so (by R10) the *Loan* and *Borrower* entities are related. The relationship type is (1:n) because a process can be executed several times meaning that a *Borrower* can have several loans. From another point of view, during the *Lend a Book* business process execution, the activity *Register Loan* sends information to the *Loan* data store after the activity *Check Book Status* read information from the *Book* data store. So, by R11, *Book* and *Loan* entities are related. The relationship type is (1:n). The relation is not mandatory on the side of the *Loan* entity because the activity that writes information is only executed if the gateway condition (*is book available?*) is true.

The *Reserve a Book* business process (Figure 4.13) involves three data stores: *Borrower*, *Book* and *Reservation*. *Borrower* and *Book* were already identified meaning that only *Reservation* will be appended as an entity to the data model. The *Attendant* participant writes information in the *Reservation* data store so, by R9, the *Attendant* entity is related with the *Reservation* entity. The relationship type is (1:n). In the same business process, the *Check Book* activity reads information from *Book* and the activity executed immediately after (*Add reservation*) writes information in the *Reservation* data store. By R11, the two corresponding entities are related (*Book* and *Reservation*). The relationship type is (1:n). The relation is not mandatory on the side of the *Reservation* entity because the activity that writes information is only executed if the gateway condition (*is possible to reserve the book?*) is true.

The *Renew a Loan* business process model (Figure 4.14) involves the data stores *Loan* and *Reservation*, which are already represented in the data model.

In the *Return a Book* business process model (Figure 4.15), more exactly in the *Penalty Treatment* sub-process (Figure 4.16), a new data store, *Receipts*, is identified so, the corresponding entity must be represented in the final data model. The *Pass Receipt* activity writes information on the *Receipts* data store and sends information to the *Borrower* participant, so (by R10) the entities *Borrower* and *Receipts* are related and the relationship type is (1:n). The *Pass Receipt* activity is performed by the *Attendant* participant, so (by R9) the *Attendant* and *Receipts* are related and the relationship type is (1:n).

Because none of the activities is cyclic or “multi-instantiable”, there are no (m:n) relationship types in the resulting data model.

All identified entities (originated by participants and data stores) and the relationships identified in each process are presented in Table 4.2.

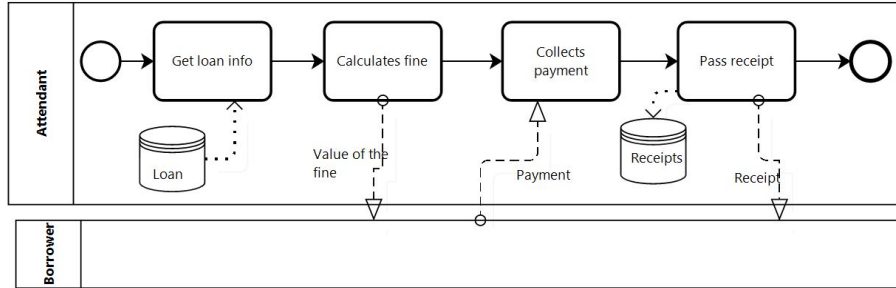


Figure 4.16: Penalty treatment business process model

Business Process	Entities	Relationships
Register User	Attendant Borrower	Attendant-Borrower(1:n), by R9
Lend a Book	Attendant Borrower Book Loan	Attendant-Loan(1:n), by R9 Borrower-Loan(1:n), by R10 Book-Loan(1:n), by R11
Reserve a Book	Attendant Borrower Book Reservation	Attendant-Reservation(1:n), by R9 Borrower-Reservation(1:n), by R10 Book-Reservation(1:n), by R11
Renew a Loan	Attendant Borrower Loan Reservation	Attendant-Loan(1:n), by R9 Borrower-Loan(1:n), by R10
Return a Book + Penalty treatment	Attendant Borrower Loan Receipt	Attendant-Loan(1:n), by R9 Attendant-Receipt(1:n), by R9 Borrower-Receipt(1:n), by R10

Table 4.2: Entities and Relationships

The resulting data model is shown in Figure 4.17.

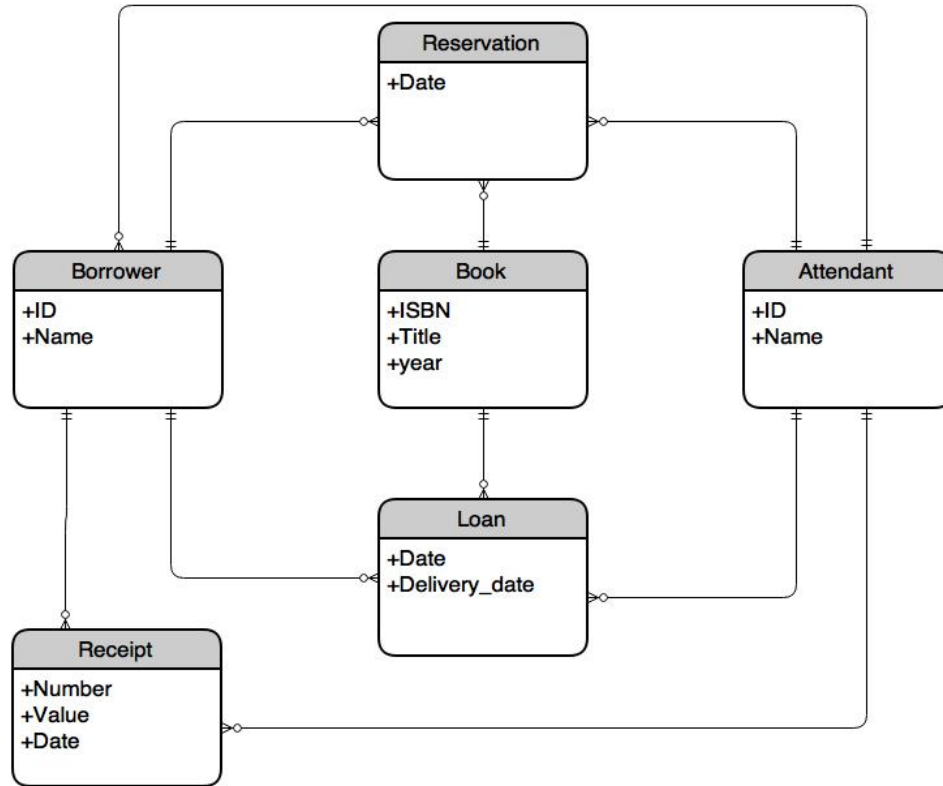


Figure 4.17: The resulting data model

4.6 Analyzing the Results

The inclusion of the data store graphical element in the BPMN most recent version, BPMN2.0 (OMG, 2011a), allows the identification of the persistent data involved in a business process. As a consequence, joining the set of business processes that will be supported by the software under development, we are capable of collecting all the information about persistent data involved in those business processes.

Analyzing the generated data model (Figure 4.17), we may say that, from a group of related business process models, we are able to generate a complete data model identifying all the entities involved, attributes and all the relationships between the entities, including optionality and cardinality.

By R12, all relationships are represented in the final data model, so in some situations, especially when a large number of business processes are involved, it may originate redundant relationships. As a consequence, the resulting data

Data Store	Process writes information	Process reads information
Borrower	Register User	Lend a Book Return a Book Renew a Loan
Loan	Lend a Book Return a Book Renew a Loan	Return a Book Renew a Loan
Book	—	Lend a Book Reserve a Book
Reservation	Reserve a Book	Lend a Book Renew a Loan
Receipt	Penalty treatment	—

Table 4.3: Entities manipulation

model should be analyzed and checked before moving on to the next step of the development process.

We mentioned previously that, usually, related business processes complement each other. To support this idea, all operations performed during the processes' execution, in what concerns persistent data manipulation, are summarized in Table 4.3.

Analyzing the data presented in Table 4.3 we may conclude that the business processes complete each other, because the information written by one process is, most of the times, used in another business process. But, we may also see that the *Book* entity (representing a data store) is not updated/inserted by any activity of the selected business processes but it is used (read) in several business process models.

Following the same reasoning, the *Receipts* entity is never used or read by any activity of the selected business processes but, during the *Penalty Treatment* sub-process execution, information is stored. The reason why this happens must be verified. In this particular case, the most probable reason is because the selected set of business processes is not complete. In fact, the *Purchase books* business process which writes in the book data store is not included in the selected set, and the information about receipts is accessed from an external accounting system that supports the organization's accounting process.

4.7 Final Remarks

It is recognized that detailed information about business processes can help to ensure that the software under development will really meet business needs (Mili et al., 2003; Giaglis, 2001; Cruz et al., 2014a). BPMN has increased its importance as a business process modeling language and it is becoming more complete. The most recent version allows identifying detailed business process, including distinguishing persistent from non-persistent data (OMG, 2011a). Thus, it becomes possible to identify data that are maintained in a persistent manner.

From a software development point of view, one of the most important, and used models, is the data model. However, to enable the ulterior obtention of the data model, it is necessary that the business process modeling is made taking data into account, i.e. the modeler must monitor the data throughout the process. Moreover, it is necessary to identify the activities that write or make use of the information stored in the data store, and ensure that the roles responsible for performing those activities are identified.

In the approach presented in section 4.2, data stores and participants in the business process model give origin to entities in the data model. The relationship between those entities is deduced from the information exchange between participants and the activities that manipulate the data stores.

The extended approach, presented in section 4.4, derives a data model based on the existing information in a set of interrelated business processes. When we are working with only one business process, the generated data model may be incomplete but, by joining together a set of interrelated business processes, we are able to get a much more complete data model because usually the information is shared by a set of related business processes, belonging to an organization. Often, the data written by a business process is used by another (or the same) business process.

The generated data model will help to ensure the alignment between business processes and the software that support them. However, it is necessary to note that if the business process models will provide the basis for the software development, they have to cover all the relevant information, including the information about the data involved in the process. The approach here presented needs highly detailed business process models especially in what concerns to data. This can make a business process model too complex and can affect its main objective, which is the description of the business process flow in a way that is understandable by all stakeholders. To serve both objectives, multiple perspectives of the model, each focusing on a specific aspect, can be created. Another solution is to develop business process modeling tools able to allow hiding/unhiding some details to represent the several views and to improve understandability of a specific aspect (Meyer et al., 2013).

It can be said that the BPMN models, if correctly created, support the automatic generation of the proper data model, serving as a basis to the software

development. The approach presented here can also be used to verify the completeness of the involved business processes (in terms of persistent data) and/or to identify possible links with other applications.

In (Weber et al., 2011) the authors conclude that, usually, business process models have bad quality. An effort to improve processes' quality and completeness is needed. The approaches presented here is a step in that direction.

Chapter 5

Deriving a Data Model from a Logical Software Architecture

One of the most difficult, and crucial, activities in software development is the identification of system functional requirements. A popular way to capture and describe those requirements is through UML use case models. During system analysis, most of this information must be incorporated into use case descriptions. A business process model identifies the activities, resources and data involved in the creation of a product or service, having lots of useful information for developing a supporting software system. This chapter proposes an approach to support the construction of use case models based on business process models emphasizing use cases descriptions, which are created using a set of predefined natural language sentences mapped from BPMN model elements.

Transforming requirement specifications into software design models is another complex and error prone software development activity. That is the 4SRS main goal. The 4SRS method generates a logical software architecture based on a use case model. The software design usually involves several models each one representing a different perspective. One of those perspectives is the data perspective which can be modeled using a data model.

In the second part of this chapter, the 4SRS method is adapted and extended in order to generate a data model supporting the generated logical software architecture, and the elicited requirements, based on the generated use case model, and the corresponding descriptions, from business process models.

5.1 Introduction

Business process modelling is being increasingly used by organizations to detect bottlenecks, waste, and deviations and to innovate by simulating possible improvements to processes (Schmiedel and vom Brocke, 2015). Business process management focus its attention on designing and documenting business processes, in order to describe which activities are performed and the dependencies between

them (Meyer et al., 2011), having lots of useful information for starting to develop a supporting software system.

Requirements elicitation is one of the first steps in software development process and is a complex and longstanding but crucial activity to the software development process (Zowghi and Coulin, 2005). A software development team needs to understand the system context and scope before starting to plan and design a solution. Some software development processes, such as the Unified Process, use the UML to support the modeling and documentation of the entire software development process (Jacobson et al., 1999). The Unified Process starts by the business process modeling and, after that, the UML use case model is used to model software requirements (Jacobson et al., 1999). Use case models aim to capture and describe the functional requirements of a system (Hull et al., 2011; Yue et al., 2011; Gomma, 2011).

A use case model comprises a set of use case diagrams and the corresponding use case descriptions (Bittner and Spence, 2003a). There are some alternatives that can be used to describe a use case, like informal text, numbered steps, pseudo-code, among others (Cockburn, 2001). Cockburn proposes a basic use case descriptions template that includes the use case name, actors, scope, context, pre-conditions, primary success scenario, alternate scenarios, amongst others (Cockburn, 2001).

The first part of this chapter describes an approach, published and presented in (Cruz et al., 2014b), to obtain a complete use case model based on a business process model. All information existing in a BPMN model that cannot be represented as an actor or as a use case will be depicted as textual use case description. Use case descriptions are, commonly, specified in Natural Language (NL) (Fantechi et al., 2003; Cockburn, 2001). As Fantechi *et al.* say, NL is easy to understand but, at the same time, could be ambiguous, redundant and with omissions (Fantechi et al., 2003; Yue et al., 2009). However, in the approach presented herein the generated descriptions are a set of controlled sentences previously defined in NL. According to Bera and Evermann, a restricted NL aims to “reduce ambiguity, redundancy, and complexity” and makes computational NL processing more “reliable, efficient, and accurate” (Bera and Evermann, 2014) and allowing to automatize the software analysis (Yue et al., 2009).

Requirements elicitation is a key step of the software development process, but there are other key points. One of them is the transformation of the requirements specification, modeled as use cases, into software design models. This is mainly because, at this stage, the problem specification starts to be transformed into a software product solution (Bragança and Machado, 2006). That is, in fact, the main goal of the 4SRS method.

The use case model is used in requirements elicitation and specification as a means to facilitate the dialog with the customer about “what” the system is supposed to do (OMG, 2012), but not “how” the system must do it. The 4SRS method ensures the transition from user requirements, specified as use case

models, into logical software architectures (Santos and Machado, 2010; Machado et al., 2006). It employs successive transformations of the software architecture in order to satisfy the elicited requirements. A logical software architecture represents the software system main components and the relation between them, allowing to understand the organization of the system.

The 4SRS method is especially useful to make the transition from requirements modeled as use cases, to the architecture of large and complex software systems (Ferreira et al., 2012), because in these cases the possibility to lose or forget some detail is high and it is very complex to manage the use cases. The 4SRS method enables us to prevent these problems, but to do so it requires a use case model with a high detail level. For this, the decomposition triangle approach, presented in section 6.3, may be used helping in decomposing and refining use cases to achieve a high detail.

After presenting an approach to derive a use case model, including use case descriptions, based on the information available in business process models, we will present another approach to use this generated use case model as input to the 4SRS leading the logical software architecture.

The second approach presented in this chapter (section 5.4) adapts and extends the 4SRS in order to generate the data model based on the derived logical software architecture (Cruz et al., 2016). This way, it will be possible to generate the software system logical architecture and its supporting data model. This second approach is presented and published in (Cruz et al., 2016).

The remainder of this chapter is structured as follows. Section 5.2 describes our approach for deriving a use case model from one business process model, specially focused in use case descriptions. Section 5.3 presents the application of this approach to a demonstration case. In section 5.4, the 4SRS method is adapted to deal with use case models using a structured and controlled language in use case descriptions and is extended to generate the data model from the derived logical software architecture. The application of this approach is illustrated through a demonstration case, integrating a set of business process models, in section 5.5. Finally, conclusions are presented in section 5.6.

5.2 From Business Process Models to a Use Case Model

It is recognized that the software that supports the business must be aligned with the business processes (Giaglis, 2001). Therefore, it is natural to try an approximation between business process modeling and software modeling. Requirements elicitation is usually the first phase on a software development process.

Shishkov *et al.* states that deriving use case models from business analysis models would be useful, since both reflect behavior within business/software

systems (Shishkov et al., 2002). Several authors already propose approaches to derive use cases from business process models (see section 3.4). All surveyed existing approaches obtain a use case diagram based on a business process model, but no one presents a proposal for obtaining the use cases description. Nevertheless, the use cases descriptions are one of the most important components of the use case model (Cockburn, 2001; Bittner and Spence, 2003b). Moreover, without descriptions most information presented in a business process model will be lost when generating the use case diagram from a business process model.

Cockburn emphasizes the use case descriptions. In Cockburn's opinion the use case writers should spend their time and effort on use case descriptions (Cockburn, 2001). The use case descriptions can specify all information needed. But, how should the use cases be written? Cockburn advises the use case writers to use sentences with a simple structure, which should be "easy to read and follow" (Cockburn, 2001) and describes a semi-formal structure to use cases description.

The CREWS (Co-operative Requirements Engineering With Scenarios) team proposes two sets of guidelines to be used on use case descriptions: six guidelines related to style and eight related to content (Rolland and Achour, 1998). Karl Cox also presents a set of structure guidelines for use case descriptions (Cox, 2002). More exactly he proposes the *CP Use Case Writing Rules*, a small set of guidelines derived from the 7 C's (Coverage, Cogent, Coherence of logic, Consistent abstraction, Consistent Structure, Consistent Grammar, Consideration of alternatives) (Cox, 2002).

Comparing CREWS and CP guidelines, the CP guidelines number is smaller and intends to be easier to apply than CREW guidelines (Phalp et al., 2007). Both provide improvements on use case descriptions quality (Phalp et al., 2007) and subsequently improve the understanding between stakeholders.

Graphically a use case diagram is very simple because it only involves actors and use cases (stickman's and ellipses with a brief description). A BPMN process diagram is graphically more complex because it involves lots of graphical elements (activities, events, gateways, data objects, pools, etc.). However a use case model can represent as much information as a BPMN model, but most of the information must be embodied in use case descriptions. So, the approach presented here is specially focused on use case descriptions for which we present a template.

The approach is divided in two main parts. First we present a set of rules to obtain a use case diagram from a BPMN model. Then we address the rules to derive the description of the uses cases previously identified.

Use case diagram generation

The presented approach is based on the private business process, where messages exchanged with other participants, or business partners, shall be represented. The proposed approach is based on the following considerations:

- The information about the participants in the process is relevant to the process, so all participants involved in messages exchange must be represented.
- An activity represents some work performed within a business process. An activity may be atomic, usually represented as a task, or non-atomic, represented as a sub-process. To avoid information loss during the application of the proposed approach, the sub-processes must be expanded.
- A manual task is a task performed without any information technology involvement (Allweyer, 2010). Nevertheless, the information about the task execution, like start and ending time or amount of resources produced and consumed, can be useful to the process monitoring to support and evaluate future decisions or improvements.

We agree with Rodriguez *et al.* on mapping a participant to an actor and one activity to a use case (Rodríguez et al., 2007). Accordingly, the rules to generate the use case diagram are explained below:

- R1: A role played by a participant (represented by a lane or a pool) must be represented by an actor in the use case diagram. The actor name is the participant name.
- R2: A lane can be the sub-division of a pool or a sub-division of another lane. These subdivisions form the actors' hierarchy:
 - If the lane is a sub-division of a pool then the actor that represents the lane is a specialization of the actor that represents the pool;
 - If the lane is a sub-division of another lane then the actor that represents the internal lane is a specialization of the actor that represents that lane.
- R4: Each activity will be represented as a use case in the use case diagram. The use case name (brief description of the action) is the activity name.
- R5: An actor that represents a pool (or a lane) is related with all use cases representing the activities that belong to the pool (or lane).
- R6: The actor that represents the participant that sends (or receives) a message to an activity is related to the use case that represents that activity.

Next subsection applies the described rules to the Nobel Prize example.

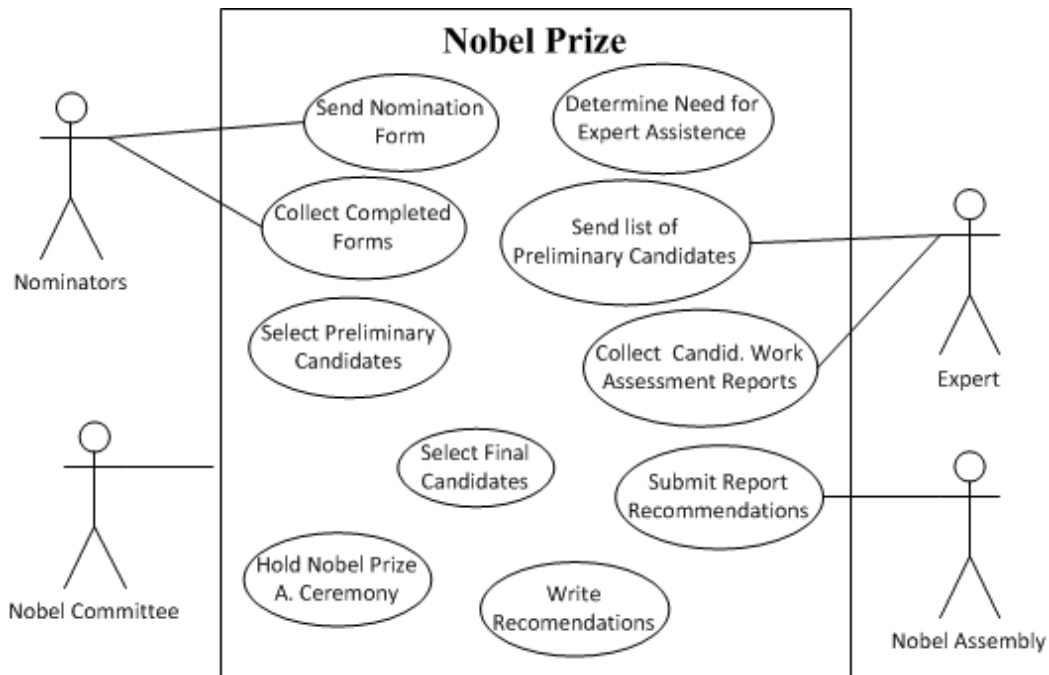


Figure 5.1: The Nobel Prize Use Case Diagram

Use case diagram generation applied to the Nobel Prize demonstration case

The diagram shown in Figure 4.4, represents the Nobel Prize BPMN Process Diagram. The presented BPMN model comprises ten activities, consequently (by rule R4 above) there will be ten use cases on the generated use case diagram. Four pools are involved in the process: *Nobel Committee*, *Nominators*, *Expert* and *Nobel Assembly*. By R1 the obtained use case diagram will have four actors with the corresponding names. The obtained Nobel Prize use case diagram is shown in Figure 5.1.

As can be seen in Figure 4.4, all activities are performed by *Nobel Committee* participant, so, by R5, all use cases are related with *Nobel Committee* actor. The *Nominators* participant sends a message to *Send Nomination Form* activity, so, by R6, the *Nominators* actor is related with the *Send Nomination Form* use case. The *Collect Completed Forms* activity receives a message from the *Nominators* pool, so, by R6, the *Nominators* actor is related with the *Collect Completed Forms* use case. The explanation for the other relationships is similar.

Getting use case descriptions

This subsection addresses the generation of use case descriptions from a private business process model. We define a template to represent a use case description

Use Case name	The use case name identifies the goal as a short active verb phrase.
Actors	List of actors involved in the use case
Pre-Conditions	Conditions that must hold or represent things that happened before the use case starts.
Post-Conditions	Conditions that must hold at the conclusion of the use case.
Trigger	Event that starts the use case.
Scenario	Sequence of interactions describing what the system must do to move the process forward.

Table 5.1: The template for describing use cases

which is a simplification of the template presented by Cockburn in (Cockburn, 2001). The proposed template is composed by six fields, which are named and described in Table 5.1.

Cockburn says that a real big and complex system can be modeled with only seven use cases (Cockburn, 2001). This yields very complex use cases with several alternative scenarios. Our approach, by transforming each BPMN activity into a different use case, yields much simpler use cases, each with a single scenario. For that reason the proposed template only attend to one (main) scenario. Pre-conditions, triggers and post-conditions enable the representation of the process flow in the use case model.

The main elements involved in a process are participants (pool and lanes), activities, gateways, events, messages, data objects, data stores and artifacts (OMG, 2011a). These elements are connected by connecting objects (sequence flow, message flow, associations and data associations). The approach being presented intends to transform business process elements, and their associated information, in a controlled set of sentences in NL, following the CREWS guidelines.

The activity name is the use case name in the use case template. The related pools or lanes represent the actors related with the use case in the use case template, as described previously.

Focusing our attention on a use case, all incoming connections and outgoing message flows, data associations, and sequence flows to events of the corresponding activity must be reflected in the use case descriptions, fulfilling the use case template previously defined.

Sequence flows outgoing an activity to a gateway or to another activity do not create a sentence in the source activity description because these connections already create sentences in the activity that receives the sequence flow.

Each connecting object makes a connection between a source (`sourceRef`) and a target (`targetRef`). Different connecting objects connect different elements. The next sub-sections describe how incoming and outgoing connections of an activity are represented in the corresponding use case template.

Data Associations

Data associations are used to move data between data objects (or data stores) and activities (OMG, 2011a). The data (physical document or information) that are created, manipulated, and used during the execution of a process are represented as data objects (or data object references) or as data stores (or data store references). A data object reference is a way to reuse data objects in the same diagram (OMG, 2011a). The same happens with the data store reference.

The sentences generated by data associations and associated data objects, or data stores, are represented in Table 5.2. The sentences will be appended to the scenario of the use case description of the use case that represents the activity.

Association

An association is used to link text annotations and other artifacts with other BPMN graphical elements (OMG, 2011a). When an association links a text annotation with an activity, the text is transcribed to the scenario of the use case that represents the activity. The text remains the same. When an association links a text annotation to a gateway, or to a sequence flow, the text is transcribed to the scenario of the use case that represents the target activity.

Message Flow

A message flow connects two pools representing the message exchange between the two participants (OMG, 2011a). A message represents the content of a communication between two Participants (OMG, 2011a). A Message is graphically represented as an envelope as we can see in Figure 4.4. The sentences originated by a message flow are described next as two different rules (MR1 and MR2).

- MR1: When an activity receives a message (message input), the use case that represents the activity will have the following sentence in its use case scenario: **Receives** <message name> [**with** <messageRef>] **from** <participant name>.
- MR2: When an activity sends a message (message output), the use case that represents the activity will have the following sentence in its use case sce-


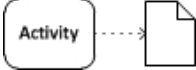

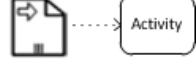
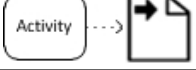
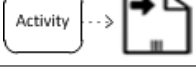


Data	Graphical representation	Originated sentence in use case scenario.
Data Object as data association source		Receives <data object name>.
Data Object as data association target		Sends <data object name>.
Data Input		Receives <data object name>.
Data Input Collection (Input set)		Receives a collection of <data object name>.
Data Output		Sends <data object name>.
Data Output Collection (Output set)		Sends a collection of <data object name>.
Data Store as data association source		Reads information from <data store name>
Data Store as data association target		Writes information on <data store name>

Table 5.2: The use case sentences originated by Data Associations

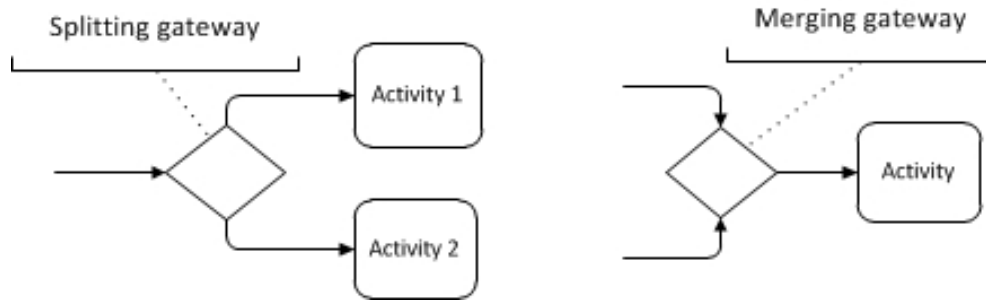


Figure 5.2: splitting and merging gateways

nario: **Sends** <message name> [with <messageRef>] to <participant name>.

MessageRef defines the message that is passed via message flow. It can be any kind of information exchanged between different pools (an email, a phone call, a document, etc.).

Sequence Flow

A sequence flow is used to show the order that activities are performed in a process (OMG, 2011a). A sequence flow can connect activities, events and gateways (OMG, 2011a).

- When a sequence flow connects two activities, it originates the next sentence as pre-condition in the use case that represents the target activity: **The <source activity name> has been completed.**
- When one activity has several incoming sequence flows, the originated sentence is the same as the sentence originated by Inclusive merging gateway (Table 5.3).

Everything that occurs between two activities must be registered in the target activity description. Involved gateways and events are treated in the next subsections.

Sequence Flow and Gateways

Gateways are used to control how the process flows, by diverging (splitting gateways) and converging (merging gateways) sequence flows. Splitting gateways have one incoming sequence flow and two or more outgoing sequence flows. Merging gateways have two or more incoming sequence flows and one outgoing sequence flow (OMG, 2011a), as we can see in Figure 5.2.

The gateway's outgoing sequence flows may have a *Condition* that allows to select alternative paths. Each outgoing sequence flow originates a sentence

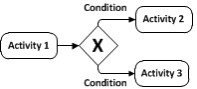
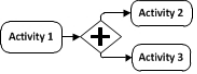
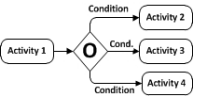
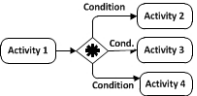
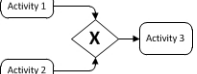
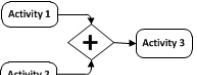
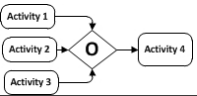
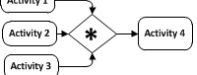
Gateway	Graphical representation	Originated Pre-condition .
Exclusive Decision		The <gateway condition> is <sequence flow condition>.
Parallel splitting		The <source name> has been completed.
Inclusive Splitting		The <sequence flow condition> is true.
Complex Splitting		The <sequence flow condition> is true.
Exclusive merging		The <source name> [exclusive or <source2 name>] has been completed.
Parallel join		The <source name> [and < source2 name>] has been completed.
Inclusive merging		The <source name> [or <source2 name>] has been completed.
Complex merging		The <source name> [or <source2 name>] has been completed.

Table 5.3: The use case pre-condition originated by gateways

represented as a pre-condition in the use case description of the sequence flow target activity. The generated sentences are represented in Table 5.3.

Sequence Flow and Events

An event is something that happens during the course of a process and that affects the process's flow (OMG, 2011a). These events usually have a cause or produce an impact (OMG, 2011a). In BPMN 2.0 there is a large number of event types, so we present a general overview of the generic sentences originated in the use case template by the different events categories (see Table 5.4).

The identified categories are grouped in four tables to address differences that can exist between sentences generated by the events of these groups of categories.

Event type category	Generic sentence originated in use case template
Start	Trigger: The <event name - event definition> occurred.
Start (Sub-Process) Interrupting	Trigger: The event <event name - event definition> occurred.
Start (Sub-Process) Non-Interrupting	Trigger: The event <event name - event definition> occurred.
Intermediate Catching	Trigger: The <event name - event definition> is received.
Intermediate Boundary Interrupting	Scenario: If the <event name - event definition> occurs, the <activity name> is interrupted.
Intermediate Boundary Non-Interrupting	Scenario: The <event name - event definition> occurred.
Intermediate Throwing	Post-condition: The <event name - event definition> is created.
End	Post-condition: The <event name - event definition> is created. The process ends.

Table 5.4: Generic sentences originated by events

Catching Event	Originated sentence in use case trigger .
None	The event <event definition> occurs.
Message	The message <event definition> arrives from <source>.
Timer	The time-date <event definition> is reached.
Escalation	The <Name - Event Definition> occurs.
Conditional	The condition <expression> become true.
Signal	The signal <event definition> arrives.
Multiple	The <event definition> [or <event definition>] occurs.
Parallel Multiple	The <event definition> [and <event definition>] occurs.

Table 5.5: The sentences originated by Start and Catching events

Sentences originating a Trigger in use case description are presented in Table 5.5. Sentences originating a Pos-condition in use case description are presented in Table 5.6. Sentences appended to the Scenario of the use case are presented in Table 5.7 and in Table 5.8 .

The events affect the sequence or the timing of the process's activities. There are three types of events: Start, Intermediate and End. Start events indicate where a process (or a sub-process) will start. End events indicate where a path of a process will end. Intermediate events indicate where something happens somewhere between the start and end of a process (OMG, 2011a).

Some events are prepared to catch triggers. These events are classified as catching events. Events that throw a result are classified as throwing events (OMG, 2011a). All start events and some intermediate events are catching events (OMG, 2011a). The sentence originated by a catching event is included as a trigger in the description of the use case that represents the activity that is started by the event. Catching events are represented as triggers because this events cause the start of the activity. The sentences originated by the Start and Catching events are detailed in Table 5.5.

All End events and some Intermediate events are Throwing events (OMG, 2011a). The sentences originated by the End and Throwing events are included as a post-condition in the description of the use case that represents the activity that throws the event. Throwing events are represented as a post-condition because

Throwing Event	Originated Pos-condition
None	The Event <Name - Event Definition> is created
Message	The message <Name - Event Definition> is sent to <participant name>.
Error	The error <Name - Event Definition> is created.
Escalation	The <Name - Event Definition> is created.
Cancel	The <Name - Event Definition> is created. The transaction is cancelled.
Compensation	The <Name - Event Definition> is created.
Signal	The signal <Name - Event Definition> is created. A Signal is broadcasted.
Terminate	The <Name - Event Definition> is created. All Activities in the Process end.
Multiple	The <Name - Event Definition> [and <Name - Event Definition>] are created.

Table 5.6: The use case descriptions originated by End and Throwing events

the event is a consequence (or a result) of the activity execution. The sentences originated by the End and Throwing events are detailed in Table 5.6.

Some events can also be classified as interrupting or non-interrupting events. Interrupting events stop its containing process whenever the event occurs. When Non-Interrupting events occur its containing process is not interrupted (OMG, 2011a). The sentences generated by Intermediate Interrupting events, originating a sentence in use case scenario, are presented in Table 5.7.

The sentences generated by Intermediate Non-Interrupting events, originating a sentence in use case scenario, are presented in Table 5.8.

An event can be thrown by an activity and caught by another. In this case the event originates a sentence in the post-condition of the use case representing the activity that throws the event and another sentence in the trigger of the use case representing the activity that catches the event.

In a use case scenario, several sentences may be generated from BPMN model elements. The sentences to be appended to the use case scenario must be according to the following order:











Intermediate Interrupting Event	Graphical representation	Originated sentence in use case scenario
Message		The <activity name> is interrupted when the message <Name - Event Definition> is received from <participant name>.
Timer		The <activity name> is interrupted when the time <Name - Event Definition> is reached.
Error		The <activity name> is interrupted when the error <Name - Event Definition> occurred.
Escalation		The <activity name> is interrupted when the <Name - Event Definition> occurred.
Cancel		The <activity name> is interrupted when the <Name - Event Definition> occurred.
Compensation		The <activity name> is interrupted when the <Name - Event Definition> occurred.
Conditional		The <activity name> is interrupted when the condition <Name - Event Definition> become true.
Signal		The <activity name> is interrupted when the signal <Name - Event Definition> arrived.
Multiple		The <activity name> is interrupted when the <Name - Event Definition> [or <Name - Event Definition>] is received.
Parallel Multiple		The <activity name> is interrupted when the <Name - Event Definition> [and <Name - Event Definition>] are received.

Table 5.7: The use case descriptions originated by Intermediate Interrupting events








Intermediate Non-Interrupting Event	Graphical representation	Originated sentence in use case scenario
Message		The message <Name - Event Definition> is received from <participant name>.
Timer		The time <Name - Event Definition> is reached.
Escalation		The <Name - Event Definition> occurred.
Conditional		The condition <Name - Event Definition> become true.
Signal		The signal <Name - Event Definition> arrived.
Multiple		The <Name - Event Definition> [or <Name - Event Definition>] is received.
Parallel Multiple		The <Name - Event Definition> [and <Name - Event Definition>] are received.

Table 5.8: The use case descriptions originated by Intermediate Non-Interrupting events

Use Case name	Send Nomination Form.
Actors	Nobel Committee, Nominator
Trigger	The time-date September is reached.
Scenario	Around 3000 invitations confidential nomination forms are sent to selected Nominators. Reads information from Nominators. Sends the Nomination Invitation to Nominator.

Table 5.9: Send Nomination Form use case description

1. All incoming connections representing messages, associations, etc. received by the activity.
2. All incoming connections representing data read by the activity.
3. All outgoing connections representing data written by the activity.
4. All outgoing connections representing messages, etc. sent by the activity.

In the next section the defined approach is applied to the Nobel Prize demonstration case.

5.3 Getting Use Case Descriptions of the Nobel Prize Demonstration Case

To present herein we select the use cases that cover a greater number of application scenarios.

As we can see in Figure 4.4, the *Send Nomination Form* activity has four incoming connections: a sequence flow from an event, giving origin to a sentence in use case trigger (Table 5.5), an incoming message flow, a data association and an association, each one generating a sentence in use case scenario. The corresponding use case descriptions are presented in Table 5.9.

The *Send List of Preliminary Candidates* activity has two incoming connections: a sequence flow from a gateway, giving origin to a pre-condition (Table 5.3) and a data association giving origin to a sentence in use case scenario. The activity also has an outgoing message flow to *Expert* participant generating a sentence in use case scenario (Table 5.2). The corresponding use case descriptions are presented in Table 5.10.

Use Case name	Send List of Preliminary Candidates.
Actors	Nobel Committee, Expert
Pre-condition	The Expert Assistance Required? is Yes.
Scenario	Reads information from Preliminary Candidates. Sends the List of Candidates to be Assessed to Expert.

Table 5.10: Send List of Preliminary Candidates use case description

Use Case name	Submit Report Recommendations.
Actors	Nobel Committee, Nobel Assembly
Pre-condition	The Write Recommendations is complete.
Scenario:	Receives information about Report with recommendations from Write recommendations. Sends the message Report with Recommendations to Nobel Assembly.

Table 5.11: Submit Report Recommendations use case template

The *Submit Report Recommendations* activity has an incoming sequence flow from another the activity *Select Final Candidates*, giving origin to a pre-condition (Table 5.3) and an incoming data association from the data object *Report with recommendations*. The activity also has an outgoing message flow to *Nobel Assembly* giving origin to a sentence in the use case scenario. The corresponding use case descriptions are presented in Table 5.11.

All use case descriptions from the Nobel committee example are summarized in Table 5.12. All identified use cases are numbered using the tag=value UML mechanism integrating the approach presented in previous section with the decomposition triangle approach (presented in next chapter, section 6.3). The Nobel committee example will be used throughout the next section as a running example.

The next section describes an approach to obtain a data model by adapting and extending the 4SRS method.

5.4 Adapting and Extending the 4SRS to Derive a Data Model

The data model is used to structure the knowledge about a specific domain and is a way to leverage the elements (or concepts) of most interest on that domain (Evans, 2011). It represents the key concepts of the problem and the relationships between them. The key concepts are also called entities.

In the approach presented herein we intend to generate the data model based on the information we have in a use case model, where each use case description is created using the set of structured Natural Language sentences defined previously in section 5.2.

The 4SRS method allows obtaining the architectural elements (system level objects) of the system based on the use case names and descriptions (Fernandes et al., 2006; Machado et al., 2006). The original 4SRS method is organized in 4 steps transforming use-case models into architectural elements (Fernandes et al., 2006; Machado et al., 2006). The 4SRS method executes a series of validations and adjustments to the original use case model.

In order to deal with structured sentences and allowing the generation of a data model, the original 4SRS steps are adapted as presented next:

- Step 1 - *Architectural element creation*: in this step, the original 4SRS method proposes the creation of three types of objects for each use case: one interface, one data and one control. However, we are able to distinguish between persistent from non-persistent data, as it happens in the BPMN language (OMG, 2011a). Following this idea, the 4SRS is adapted to also distinguish persistent data from non-persistent data, by creating two different types of elements involving data: persistent data and volatile data.

Use case Name	Use case Description
{U1} Send Nomination Form	Actors: Nobel Committee, Nominator Trigger: The time-date September is reached. Scenario: Read information from <Nominator>. Around 3000 invitations confidential nomination forms are sent to selected Nominators. Sends the Nomination Invitation to <Nominator>.
{U2} Collect Completed forms	Actors: Nobel Committee, Nominator Pre-condition: Send Nomination Form has been completed. Scenario: Receives nomination Form from <Nominator>. Write information on <Nominator>.
{U3} Select Preliminary Candidates	Actors: Nobel Committee Pre-condition: Collect Completed forms has been completed. Scenario: Read information from <Nominator>. Write information on <Candidates>.
{U4} Determine need for Expert	Actors: Nobel Committee Pre-condition: Select Preliminary Candidates has been completed.
{U5} Send List of Preliminary Candidates	Actors: Nobel Committee, Expert Pre-condition: The Expert Assistance Required? is Yes. Scenario: Read information from <[Preliminary] Candidates>. Sends the List of Candidates to be Assessed to <Expert>.
{U6} Collect assessment Report	Actors: Nobel Committee, Expert Pre-condition: Send List of Preliminary Candidates has been completed. Scenario: Receives assessments from <Expert>. Write information on <Assessment>.
{U7} Select Final Candidates	Actors: Nobel Committee Pre-condition: Collect assessment Report has been completed. Scenario: Read information from <Assessment>.
{U8} Write Recommendations	Actors: Nobel Committee Pre-condition: The Expert Assistance Required? is No OR Select Final Candidates has been completed. Scenario: Write information on <Recommendations>.

Table 5.12: The descriptions of the use cases using the defined template (part I)

Use case Name	Use case Description
{U9}Submit Report Recommendations	Actors: Nobel Committee, Nobel Assembly Pre-condition: Write Recommendations has been completed. Scenario: Read information from <Recommendations>. Sends Recommendations to <Nobel Assembly>.
{U10}Hold Nobel Prize ceremony	Actors: Nobel Committee Pre-condition: Submit Report Recommendations has been completed. Pos-condition: The process ends.

Table 5.13: The descriptions of the use cases using the defined template (part II)

Each element is labeled with the name of the use case followed by the appropriate type: *i* (interface), *c* (control), *dp* (data persistent) and *dv* (data volatile). This step can be automated since it does not involve decisions.

- Step 2 - *Architectural element elimination*: based on the textual description of each use case, it is necessary to decide which of the four elements, created in step 1, must be maintained. This step allows detecting and eliminating redundancy in requirements. This step is divided into seven micro-steps:
 - Step 2i - *Use cases classification*: In this micro step each use case is classified as interface, data persistent, data volatile, control, or any combination of these. This classification aims to facilitate the transformation of each use case in architectural elements as it provides clues about which categories of elements to use and how they are related. Since we are dealing with a set of structured sentences in use case descriptions, it is possible to classify the use case following the suggestions:
 - * When a use case exchanges information with something or someone (usually an actor), the *i-interface* type must be selected. One use case represents the interaction with an external participant (represented as an actor) using sentences like *Receives <message name> from <actor name>* or *Sends <message name> to <actor name>*.
 - * When information is stored or retrieved then the type *dp-data persistent* must be selected. Information is stored or retrieved when sentences like *Writes information on <data store name>* or *Reads information from <data store name>* are part of the use case description.

- * When in a use case description we have sentences like *Receives* $\langle document\ name \rangle$ or *Sends* $\langle document\ name \rangle$, this means that we are dealing with non-persistent data, thus the type *dv-data volatile* must be selected. It is natural to find non-persistent data transformed in persistent data because in a business process, most of the times, data received from a participant is then stored in a data store.
- * When a use case description has a trigger, a pre-condition or a post-condition, then the type *c-control* must be selected. Besides that, some use case names clearly identifies the use case as a c-control element like for example use case with name begin by *Check, Verifies, etc.*

Taking as example the $\{U6\}$ - *Collect assessment Report* use case (from Table 5.12) we may see that the use case is receiving non-persistent data (assessments) from expert (*Receives assessments from Expert*), meaning that it is interacting with someone, so this is classified as *i-interface* and as *dv-data volatile*. Reading the use case descriptions we may also see that we are dealing with persistent data because the use case *Writes information on* $\langle Assessment \rangle$, so the use case is also classified as *dp-data persistent*. The use case has to control the use case pre-condition (*Send List of Preliminary Candidates has been completed*). So, the use case is also classified as *c-control*. Summarizing, this use case will be classified as *c, i, dv and dp*.

- Step 2ii: *Local elimination* - the purpose of this micro step is to check if each architectural element created in step 1 makes sense for the problem domain. Those that do not make sense should be eliminated.
- Step 2iii: *Architectural element naming* - Each architectural element created, should receive a name fitting its original use case as well as the role that it has in the system.

Taking as example the $\{06-dp\}$ - *Collect assessment Report* persistent data element created in $\{U6\}$ - *Collect assessment Report* use case, we may rename it to $\{06-dp\}$ - *Write Assessment* to better fit the element purpose.

- Step 2iv: *Architectural element description* - Each architectural element that received a name in previous micro-step should be described according to the corresponding system requirements, in order to be included in the logic model (depicted by objects diagram).

Taking as example the $\{06-dp\}$ - *Write Assessment* element created in $\{U6\}$ - *Collect assessment Report* use case, the corresponding description may be the sentence belonging to the original description which leads to the element classification as *dp-persistent data*. The

sentence is *Writes information on Assessment*. The other sentences will lead to the other types of elements. This is a manual step, so the software architect may complement the description with other information every time it is necessary. At this software development stage, the stakeholders are still involved in the process so they can provide useful information to complement the information generated based on business process models.

- Step 2v: *Architectural element representation* - This micro-step, through an analysis of each element, ensures the semantic consistency of the logic model, detects and eliminates redundancy, and enables the discovery of anomalies in use case models, namely missing requirements. Taking as example the $\{U1\}$ -*Send Nomination Form* and the $\{U3\}$ -*Select Preliminary Candidates* use cases, we may see that both have the same *dp* element (*dp-Read Nominator*) because both are reading information from \langle Nominator \rangle . As such, both elements can be represented by the same architectural element (*dp-Read Nominator*).

- Step 2vi: *Global elimination* - In this step all micro architectural elements that are represented by other architectural elements are eliminated, since the requirements that correspond to these architectural elements no longer belong to them.

Continuing the example presented in previous micro-step where we conclude that the $\{01.dp\}$ *Read Nominator* can be represented by $\{03.dp\}$ *Read Nominator* architectural elements or vice-versa, one of them must be eliminated.

- Step 2vii: *Architectural elements renaming* - This micro step aims to rename all the remaining architectural elements.

Sometimes, when we have one architectural element, representing several architectural elements (micro-step 2v), the name of this element may be renamed to better fit its purpose.

- Step 3: *Architectural elements aggregation and packaging* - the architectural elements that remain after the elimination, and those in which it is possible and exist advantages in their unification, are aggregated; At this step, architectural elements that have similar characteristics and can be treated in an unified way are aggregated in the same package.

In the Nobel Prize example, the architectural elements representing persistent data manipulation can be package in *P3 - Database Control*. Architectural elements representing controlling actions are packaged in *P2 - Business rules* and elements representing user interface are packaged in *P1 - User Interface*. The use case model facilitates the identification of the system functionalities provided to a specific type of user or external system

(actor) (Tiwari and Gupta, 2015) by identifying the relations between the actors and use cases. This way, the *P1 - User Interface* package may be divided in several sub-packages representing interactions with specific actors. As such, architectural elements representing interactions with *Nominator* actor are packaged in *P1.1 - UI Nominator*. Architectural elements representing interactions with *Expert* actor are packaged in *P1.2 - UI Expert* and architectural elements representing interactions with *Nobel Assembly* are packaged in *P1.3 - UI Nobel Assembly* (see Figure 5.4).

- Step 4: *Architectural elements association* - associations must link the elements resulting from the aggregation based on use cases textual descriptions.

The 4SRS creates a tabular transformation to monitor all the steps. An example of the table can be seen in Figure 5.3.

Step 1 - element creation	Step 2 - object elimination							Step 3 - object packaging & aggregation
Element name	2i - use case	2ii - local	2iii - object naming	2iv - object description	2v - object	2vi - global	2vii - object renaming	
{U1} Send Nomination Form	i c dp dv							
{O1.i}		1	Send Nomination Form	Around 3000 invitations confidential nomination forms are sent to selected Nominators.			Send Nomination Form	P1.1 - UI Nominator
{O1.c}		1	Controls start	Trigger: The time-date <September> is reached.			Check date	P2 - Business Rules
{O1.dp}		1	Reads nominator	Reads information from <Nominator>.			Reads nominator	P3 - Database control
{O1.dv}		1	Nomination form	Sends the Nomination Invitation to <Nominator>.			Nomination form	P1.1 - UI Nominator
{U2} Collect Completed forms	i c dp dv							
{O2.i}		1	Collect forms	Receives nomination Form from <Nominator>.			Collect forms	P1.1 - UI Nominator
{O2.c}		1	Control Nominations form	Pre-condition: Send Nomination Form has been completed.			Check nomination forms	P2 - Business Rules
{O2.dp}		1	Writes Nominator	Writes information on <Nominator>.			Writes Nominator	P3 - Database control
{O2.dv}		1	Nomination form	Receives nomination Form from <Nominator>.	{O1.dv}	kill	Collect forms	P1.1 - UI Nominator
{U3} Select Preliminary	i c dp							

Figure 5.3: The 4SRS table (an excerpt)

The logical architecture resulting from the application of the 4SRS to the Nobel Prize example, is represented in Figure 5.4.

To define a persistent data model one needs to identify the domain entities, their attributes, and the relationships ((1 : n), (m : n) or (1 : 1)) between entities (Weske, 2012). Therefore, the 4SRS will be extended with three additional steps, which are:

- Step 5: *Entities creation* - in this step, the entities involved in each use case are identified.

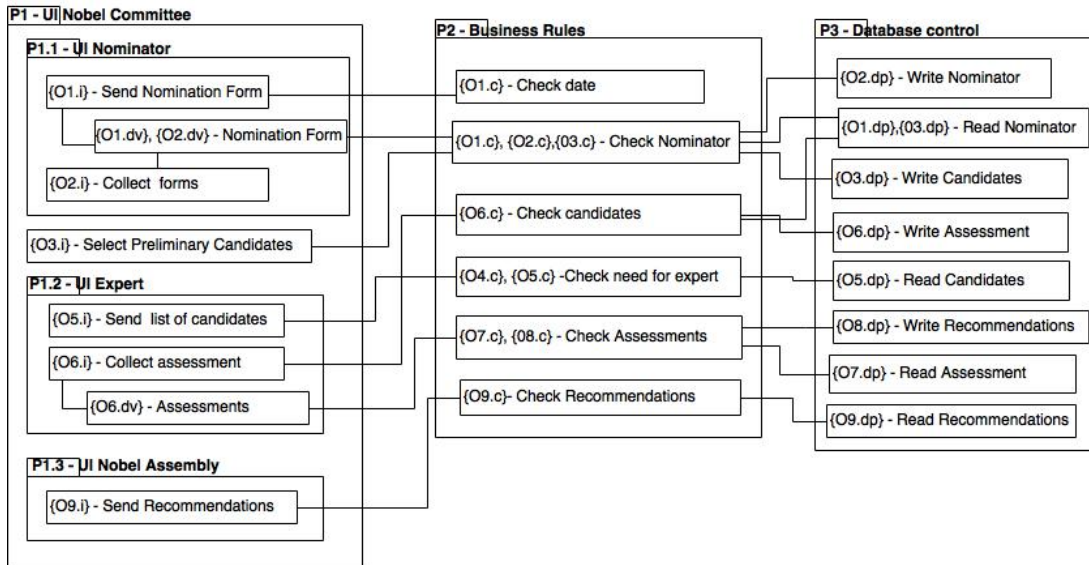


Figure 5.4: The resulting logical architecture

- Step 6: *Relationships identification* - in this step, the relationship between the entities identified in step 5 are identified.
- Step 7: *Entity attributes identification* - in this step, the attributes belonging to each entity are identified.

Each one of the steps is explained next.

Step 5 - Entities creation

An entity is something identifiable, or a concept in the real world that is important to the modeling purpose (Weske, 2012). To identify the entities this step is divided into two micro-steps, explained next:

- *Micro-step 5i: Entities identification* - Focusing on the *-dp* architectural elements remaining in the generated logical architecture, each read, written or updated element gives origin to an entity in the resulting data model.

Looking to the generated logical architecture represented in Figure 5.4, we may see that we are reading or writing information in *Nominator*, *Candidates*, *Assessment* and *Recommendations* thus, each one gives origin to an entity in the resulting data model.

- *Micro-step 5ii: Entities representation* - This step produces the final list of entities, by detecting and removing the repeated entities. It is usual to find use cases reading information that is written (or updated) by other use cases, especially when we are working with a large number of use cases. As

such, it is very common to identify (in the previous micro-step) the same entity several times. These duplicated entities are eliminated.

Backing to the Nobel Prize example and looking to the generated logical architecture (represented in Figure 5.4), in $\{O2.dp\}$ - *Write Nominator* architectural element, the entity *Nominator* is identified. The same entity is also identified in $\{O1.dp\}$ e $\{O3.dp\}$ - *Read Nominator* architectural element.

The *Candidates* entity may be identified in $\{O3.dp\}$ - *Write Candidates* and in $\{O5.dp\}$ - *Read Candidates* architecture elements. Both entities are represented by the *Candidates* entity. Similar reasoning may be used in *Assessment* and *Recommendations* entities.

Step 6 - Relationships identification

A relationship between two entities is represented through an association between those entities (Chen, 1976). The role of an entity in a relationship is the function that it executes in that relationship. A relationship between two entities can be classified according to two aspects, Cardinality and Optionality. Both terms are used to denote the number of attributes in a relation. Cardinality represents the maximum number of instances (one or many) of an entity in relation to another entity. Relationship optionality represents the minimum number of elements that exist on that side of the relationship. It may be 1 (the relation is mandatory) or 0 (the relation is not mandatory).

Focusing on the elements, remaining in the resulting logical architecture, that store information (elements with name starting by *write* or *update*), we must verify in which conditions the information is stored. When an $\{-dp\}$ - *persistent data* element is related with a $\{-c\}$ - *control* element that verifies the information about another entity, we may conclude that the information stored is related with the information checked. Usually, this verification is done by reading information already stored. As a consequence, the entity that represents the written information is related with the entity that represents the checked (and read) information. The relationship is (1:n) from the entity that represents the information checked (previously written information) to the entity that represents the written information because the same information can be read several times and associated to different written information items. On the other hand the information is stored only once.

By default the relationship is mandatory on the side of the information checked (the information must be verified) and is not mandatory on the side of the written entity because the information may be written, or not, depending on the verification result.

Looking to the $\{06.dp\}$ - *Write Assessments* element we may see that it is related with $\{06.c\}$ - *Check Candidates*, so we may conclude the *Assessments*

entity is related with the *Candidates* entity. The same happens between *Candidates* and *Nominator* and between *Recommendations* and *Assessments*. The relationship is (1:n) from the read to the written entity and not mandatory from the side of the written entity.

Step 7 - Entity Attributes identification

The information, or the properties, about an entity are expressed through a set of attributes (Weske, 2012). Since we are dealing with use cases which descriptions are generated from business process models, in some cases to prevent model complexity the properties are not identified. Nevertheless, in some cases, especially when information is stored or retrieved, the use case description may have a document in attachment (Cruz et al., 2014b) originated from attachments in BPMN data elements or messages. The document may identify items stored or retrieved. In that case, each item represents an entity attribute. In cases where the properties are not identified, the software architect may ask for more detailed information, complementing this way the information generated from the business process models.

In the 4SRS method, most of the steps are manual and some of them require the software architect expertise (Machado et al., 2006). At this software development process phase, the stakeholders are still involved and available to provide answers.

The resulting data model is shown in Figure 5.5. In the resulting data model, we are using the following syntax for each relationship end. Focusing in one side of a relationship type and considering the optionality and cardinality together we have: 0 or 1 (represented as $+o-$), 1 ($+—$), 0 to many ($-o\infty$) and 1 to many ($-—\infty$).

Through *micro-step 5i* we are able to identify the entities *Nominator*, *Candidate*, *Assessments* and *Recommendations* representing data stored.

Applying *step 6*, we are able to identify relationships between: *Nominator* and *Candidates*, *Candidates* and *Assessments*, *Assessments* and *Recommendations*. As previously explained, the relationships are (1:n) from first to the second entity and the relationship is mandatory on the side of the first entity and not mandatory on the other side.

The next section presents a second, more complete, demonstration case.

5.5 Demonstration Case Aggregating a Set of Business Process Models

In this section we use, as a second demonstration case, the School Library System where a group of five interrelated business process models have been selected. The

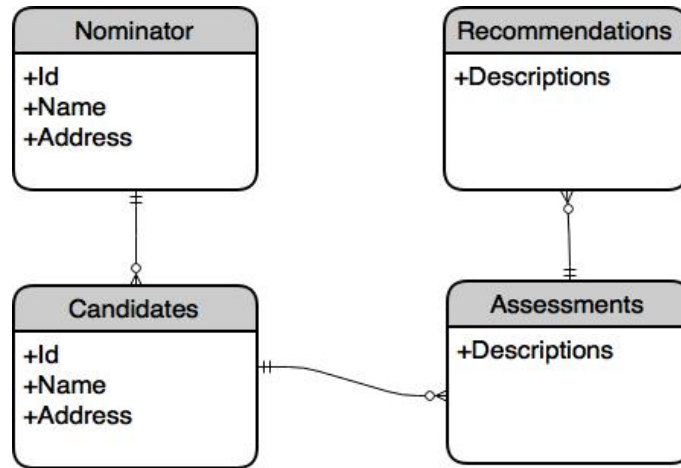


Figure 5.5: The resulting data model

selected business processes are: Register User, Lend a Book, Reserve a Book, Renew a Loan and Return a Book. The Return a Book business process model includes a sub-process, Penalty treatment. The set of business processes were aggregated in a use case model following the approach presented in chapter 6 and in (Cruz et al., 2015a).

The second approach presented in chapter 6 starts by identifying the set of business processes that will be supported by the software under development, identifying the system scope. Then a use case model, divided in several abstraction levels, is created based on a set of identified business process models. In level 1, the highest abstraction level use cases are represented in the use case model, each one representing a business process. Each use case is then decomposed and refined in a use case model in the next level. The decomposition ends when all use cases representing processes and sub-processes are decomposed into atomic activities, each one being represented as a use case. All existing information, such as data involved in the process, decisions that have to be made, exchanged messages and so on, is depicted in use cases descriptions as presented in section 5.

The resulting use case model forms a functional requirements model, that is especially prepared to be used as input of the 4SRS (4 Step Rule Set). This resulting use case model may be represented as a tree structure. The 4SRS selects all leaves from the derived tree structure obtaining, this way, the most detailed and non-redundant information we can get. The derived use case model, including use case diagram and corresponding use cases description, can be found Appendix A.

The architectural elements are aggregated in three packages: P1 - User Interface, P2 - Business Rules and P3 - Database Control.

The resulting logical architecture by applying the first four steps of the 4SRS

to the set of business processes selected from a Library system, is represented in Figure 5.6.

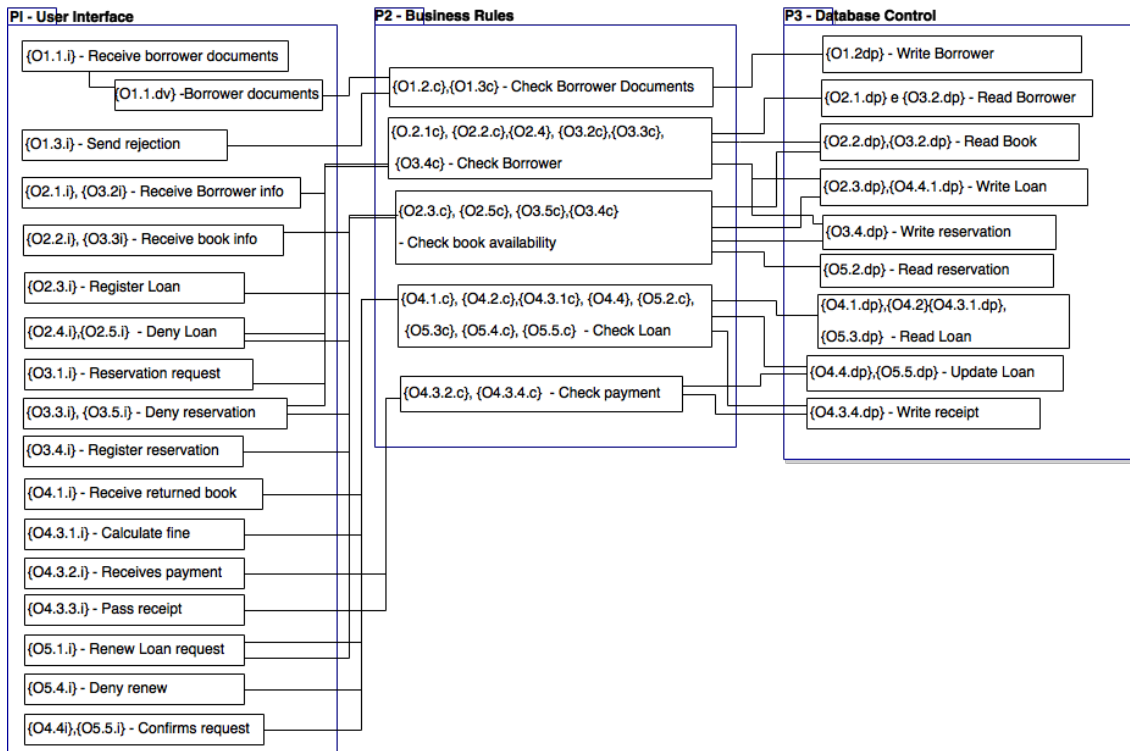


Figure 5.6: The resulting logical architecture

Analyzing the generated logical architecture we may see that, by step 5i, we are able to identify *Borrower*, *Book*, *Loan*, *Reservation* and *Receipt* as entities.

Looking to the logical architecture we may see that the *Write Loan* elements are related with *Check Borrower*. Thus, by 6i, *Loan* entity is related with the *Borrower* and the relationship is (1:n) and mandatory on the *Borrower* side and not mandatory on the *Loan* side. The same happens with *Write reservation* element and *Check Borrower*, so *Reservation* entity is related with the *Borrower* and the relationship is (1:n), mandatory on the *Borrower* and not mandatory on the so *reservation* side.

Looking to the logical architecture we may see that the *Write Loan* elements are related with *Check book availability*. Thus, *Loan* entity is related with the *Book* and the relationship is (1:n) and mandatory on the *Book* side and not mandatory on the *Loan* side. The same happens with *Write reservation* element and *Check book availability*, so *Reservation* entity is related with the *Book* and the relationship is (1:n), mandatory on the *Book* and not mandatory on the so *reservation* side.

Following the same idea, the *Write receipt* element is related with the *Check*

Loan element so, *Loan* is related with *Receipt* and the relationship is (1:n), mandatory on the *Loan* side and not mandatory on the *Receipt* side.

The resulting data model is presented in Figure 5.7.

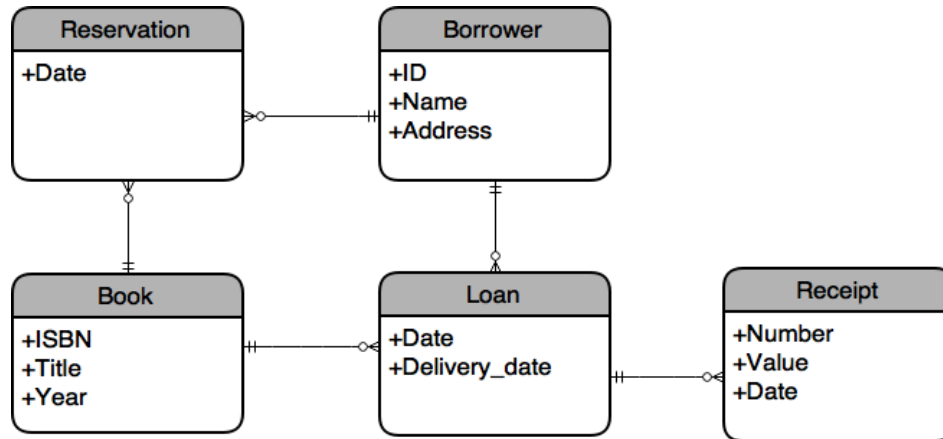


Figure 5.7: The resulting library data model

5.6 Final Remarks

This chapter starts by presenting an approach to generate a use case model, highlighting use case descriptions, from a private BPMN process diagram. This first approach starts by presenting a set of rules to generate the use case diagram in which each activity in the BPMN model gives origin to a use case and a participant gives origin to an actor in use case model. To identify the use cases description, a set of structured sentences are created in NL. Each sentence represents an incoming or outgoing connection from the use case corresponding activity.

BPMN has originally been designed to be a language easy to understand by all stakeholders involved (Magnani and Montesi, 2009; OMG, 2011a). Nevertheless, with the increase in the number of its graphical elements, in its most recent version (BPMN2.0), the language has become more complex and consequently difficult to understand (Polancic, 2015). The first approach presented herein helps understanding BPMN models, as it translates a model to NL, promoting the understanding between the involved stakeholders.

The BPMN2.0 allows business process models to be highly detailed. This is good news if one intends to use BPMN models as a basis to the development of the software that supports the business. The presented approach benefits from a detailed business process model, as greater business process detail yields a more complete use case model.

In the software development process many models are used to represent different points of view. Some models, like use case models, may have information that can be used to generate other software models. Nevertheless, most of that information is in use case descriptions, commonly specified in NL (Fantechi et al., 2003; Cockburn, 2001). NL may be easy to understand but, at the same time, can be ambiguous, redundant and with omissions (Fantechi et al., 2003). The approach presented herein is dealing with a set of controlled sentences previously defined in NL facilitating an automated analysis. The set of sentences are defined with the purpose of transforming business process models (BPMN) into use case models (Cruz et al., 2014b).

Generating a complete use case model from a business process model allows us to use existing methods, techniques and tools to generate other software models from use case models. One of those methods is the 4SRS, which generates a logical software architecture from user requirements, represented as use cases (Machado et al., 2006).

In the second approach presented in this chapter, the 4SRS is adapted and extended in order to generate a data model supporting the resulting logical architecture. Herein, we start by distinguishing persistent data from non-persistent data allowing to create three new 4SRS steps with the aim of identifying entities, the relationships between entities and the entities' attributes.

The generation of the use case model, including the use case descriptions, from business process models is prepared to be automatic as well as the construction of the 4SRS tabular transformation and its first step.

The new steps added to the 4SRS method, to generate the data model, are also automatable because despite we are working with NL, we are basing our work in a structured and controlled language which is a set of sentences previously generated from the information existing in a set of BPMN models.

Integrating the two approaches presented in this chapter and the approaches presented in chapter 6 it is possible to generate the complete use case model, the logical software architecture and the data model of the software that will support the business, based on the set of business process models helping to ensure the alignment between business process models and software models (Cruz et al., 2014a).

In a BPMN model, a sub-process may be used to describe the common part of different process models. Thus, when we are grouping a set of interrelated business process models, some use cases may appear more than once on the resulting use case model (Cruz et al., 2015a). The 4SRS method is prepared to detect and eliminate duplicated use cases (Machado et al., 2006) (step 3) making it suitable for dealing with complexity.

The approach presented here to derive a data model from use case models was specially designed to be integrated with the generation of use cases from a set of interrelated BPMN models. Nevertheless, the approach can be generalized to use case models created using the same structured language. The generated

use case descriptions can be complemented with information provided by other sources every time it is necessary.

The presented approaches enable traceability between business processes and the corresponding elements in software models.

Chapter 6

Deriving a Logical Software Architecture from Business Process Models

Organizations, and consequently the information systems that support them and that support their businesses, are becoming increasingly larger and complex. As a consequence, the models used as basis for software development are increasingly complex and hard to maintain. One of those models is the use case model, which is used in the beginning of the software development process to model the system requirements. A use case model can be created with a high abstraction level or a very detailed level. A use case model with a very detailed level can be much more useful to software development teams but, at the same time, it may become very complex and hard to understand.

This chapter starts by presenting an approach, named as decomposition triangle, to decompose and refine use cases. The decomposition triangle is an iterative and incremental approach to support the construction of UML use case diagrams as a first documentation effort of the requirements elicitation activities. The approach adopts a refinement mechanism to detail use cases, in a controlled way, as a means to obtain a functional requirements model of the system to be designed.

The decomposition triangle approach is then used to aggregate in one use case model the set of business process models that will be supported by the software under development.

6.1 Introduction

In the software development process, one of the first and crucial activities is the identification of the system scope and the understanding of what the system is supposed to do. UML use case models are one of the most popular adopted techniques to capture and describe the functional requirements of a system.

Real-world problems are, typically, complex and extensive. This can give origin to very complex use case models, especially when a high detail level is

required. In this case, the use case models could become hard to understand and to manage because the number of use cases tends to be large and the relationships between them tend to be complex. On the other hand, working with use cases with a very high abstraction level makes the verification of the correctness of the design solution a very difficult task (Berenbach, 2004).

When a system is too large, the software being developed can be divided and delivered to the customer in parts. However, in order to maintain the consistency, the system analysis and modeling shall preferably be done at one time (Berenbach, 2004).

A use case model also can be divided into several smaller models. But, in that case, some connections or relationships between those models can be lost. Besides that, keeping the consistency between them can be a problem (Hausmann et al., 2002; Regnell et al., 1996). This drives us to the question: how can we manage use cases when we are dealing with a large and complex system and when low abstraction use cases are clearly more useful to the project than high abstractions ones?

For software development teams, use cases are a way to understand the behavior of the system, serving, many times, as a basis for the functional specification of the system (Bittner and Spence, 2003a). As Dobing and Parsons say, use cases serve, many times, to guide the whole software development process (Dobing and Parsons, 2000). Indeed, some software development teams base the entire software development process on the list of requirements, identified in the development process first stage, and modeled using use case models. The use case models are used to plan all project, including scheduling the costs, the resources and the delivery dates (Issa, 2007). If the software development teams base their work on abstract and ambiguous use cases, it could imply failures on the plan and on the resulting product. On the other hand, it could be sometimes hard to work with concrete use cases because of their large number and complexity. Glinz agrees that every large system needs to be decomposed in order to make it comprehensible and manageable (Glinz, 2000).

Ideally, it should be possible to decrease and increase the abstraction level whenever it is necessary. But, how can a use case model have high abstraction level and low abstraction level at the same time? In section 6.3, we present an approach, named as the decomposition triangle, to decompose a use case model into several levels of abstractions based on a top-down structured and controlled decomposition. To do that, we need to extend the UML use case meta-model. The decomposition triangle approach is used in the second approach presented in this chapter to aggregate and merge the information we have in a set of interrelated business processes into one integrated use case model.

It is recognized that the software that supports the business must be aligned with the business processes (Rodríguez et al., 2007; Dietz, 2003; Giaglis, 2001). Shishkov *et al.* state that deriving use case models from business analysis models would be useful, since both reflect behavior within business/software systems

(Shishkov et al., 2002).

Usually an organization deals with several business processes. As a consequence, a software product does not usually support only one business process, but rather a set of business processes. This drives us to the question: “Is it possible to systematically derive a use case model from a set of interrelated business process models?”

Approaches such as BPEL (Business Process Execution Language) allow the execution of business processes in a service-oriented perspective by integrating enterprise applications (Liang et al., 2008). Such approaches, however, require the existence of services, which could be internal or external to the organization, whose orchestration can be done using BPEL. The aim of the second approach being presented in this chapter is to generate models that can be used as basis to the development of software that supports the business processes. To plan and design a suitable software supporting system, first it is necessary to know the main requirements that must be supported by the software under development. This identifies the functional requirements (representing them as a use case model) based on the set of interrelated business processes.

The 4SRS method requires a use case model with a high detail level. The two approaches we are presenting in this chapter are especially useful for preparing data (user’s requirements) to feed the 4SRS method. The decomposition triangle approach is presented and published in (Cruz et al., 2014c). This second approach is presented and published in (Cruz et al., 2015a).

The remainder of this chapter is structured as follows. In the next section, the proposal to the extension of the UML use case meta-model is presented. Section 6.3 describes our approach for use case model decomposition and presents a demonstration case. Section 6.4 explains the relation between the decomposition triangle and the 4SRS method. In section 6.5 the decomposition triangle approach is used to aggregate in one use case model all information we have in the set of business process models. The application of this second approach is illustrated through a demonstration case in section 6.6. Finally, some conclusions are presented in section 6.7.

6.2 Extending the UML 2.5 Use Case Meta-model

Use case models aim to capture and describe the functional requirements of a system (Hull et al., 2011; Yue et al., 2011; Gomma, 2011). Booch *et al.* say that use case models, when defined by Ivar Jacobson, aimed to describe the behavior of the system from the users point of view (Booch et al., 1998). So, it is expected that a use case model specifies what a system is supposed to do (OMG, 2012). In (OMG, 2012), a use case is defined as a specification of a set of behaviors

performed by an actor.

Whittle and Jayaraman (Whittle and Jayaraman, 2006) define use cases as a set of scenarios where a scenario is “an expected execution trace of a system”. In fact, one possible approach to model a system, using a use case model, starts by identifying all possible scenarios and then generalizes them in order to create the use case model (Issa, 2007).

In (OMG, 2012) a use case is defined as a *behaviored classifier* that represents a declaration of a set of offered behaviors. Each use case specifies some behavior, possibly including variants, that the subject can perform in collaboration with one or more actors.

A use case model should identify the system boundaries (marked by a rectangle) and identify the actors which are represented by a “stickman” icon outside the system boundaries (Hull et al., 2011; OMG, 2012). An actor is someone or something that interacts with the system (OMG, 2012). So, an actor is always related to one or more use cases. A use case is graphically represented by an ellipse and contains a brief description of the action (Bittner and Spence, 2003a).

A use case diagram is composed by actors and use cases. Each use case shall have an associated description and can have pre-conditions and post-conditions. There are some alternatives that can be used to describe a use case, like informal text, numbered steps, pseudo-code, among others (Machado et al., 2005).

Use cases can be related through *include* and *extend* relationships. A use case can *include* another use case and can be *extended* by other use cases (Berenbach, 2004). These relationship types are controversial (Glinz, 2000; Azevedo et al., 2010), especially the *extend* relationship because sometimes this relationship is not interpreted on the same way by different team members which leads to misunderstandings. The study presented in (Bolloju and Leung, 2006) concludes that most of the use case relationships’ errors are related with the *extend* relationship type, and some practitioners recommend not using this type of relationship.

In the approach presented in the next section we aim to create the possibility to “zoom in” to a use case to know the internal functional detail of that use case. To do that, we intend to create the possibility to refine, and decompose, a use case in a use case model, decreasing the abstraction level and, consequently, the use case ambiguity. Nevertheless, the UML 2.5 use case meta-model (defined in (OMG, 2012)) does not permit this type of relationship (between a use case and a use case model). So, the UML use case meta-model must be extended by a new *refine* relationship and by a new meta-class to represent the use case model. The UML use case meta-model extension is presented in the next subsection.

In our proposed approach we refine a use case by decomposition. When we are de-composing one use case we are dividing it into smaller parts adding new detailed information about its functionality. So, when we are decomposing a use case it must be decomposed into two or more use cases that have to be grouped together. One use case cannot decompose another use case by itself, but a group of use cases can.

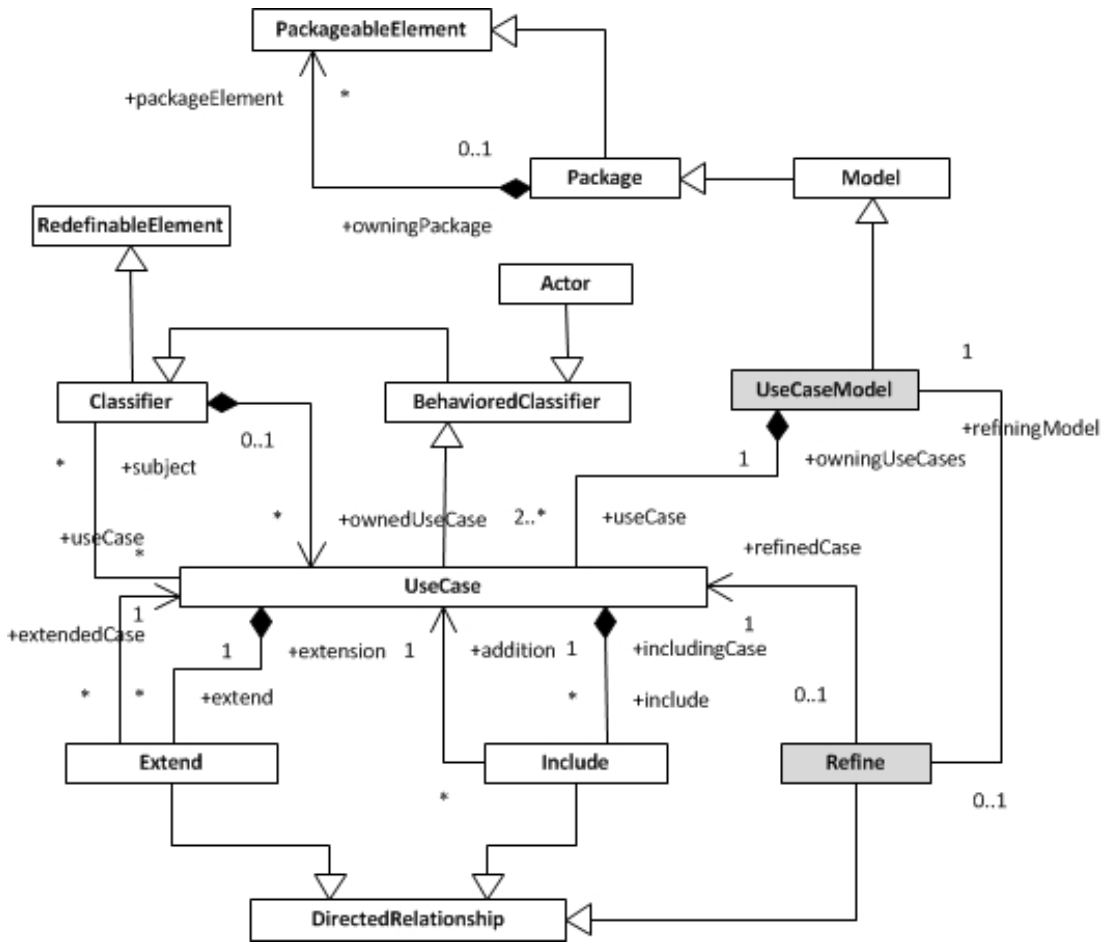


Figure 6.1: Extended UML use case meta-model

According to (OMG, 2012) a *package* is used to group elements (*PackageableElements*). As referred in (OMG, 2012) a *PackageableElement* is a *NamedElement* that may be owned directly by a *Package*. A *Package* may include *Packages*, *Dependencies*, *UseCases*, *Classes* and others.

As one can see in UML 2.5 specification, a *Model* class specializes *Package* to specify the view point that the model intended to represent (OMG, 2012). A *Model* has a set of members (*PackageableElements*) that together describe the system being modeled (OMG, 2012), so, a *Model* will group the elements used to represent a view point. A *Model* describes the relevant aspects of a system at the appropriate level of detail. Each model is meant to be complete, but models can have abstraction dependencies between them. The elements contained in one model can be refined in another model (OMG, 2012). The UML 2.5 specification do not mention if the model used to refine a model element could be the same kind and does not prohibit it either, so we assume that a model element can be refined in a model of the same kind.

We propose a new meta-class element named *UseCaseModel* to support the idea of grouping the elements used to represent the use case point of view. A *UseCaseModel* specializes *Model*, constraining the possible *PackageableElements* to use cases, actors and relationships in order to represent the use case point of view.

As represented in figure 6.1, a use case is a *BehavioeredClassifier*, which is a specialization of *Classifier* and *Classifier* is a *RedefinableElement* (OMG, 2012). This means that a use case can be redefined differently in the context of another Classifier. The *Classifier* symbol for the use case is an ellipse as a replacement for the standard rectangle notation (OMG, 2012).

A use case represents a behavior performed by one or more actors (OMG, 2012). So, each use case must be related with one, or more actors, and one actor must be related to one, or more use cases.

The proposed extension to the UML use case meta-model is represented in figure 6.1. The meta-model elements that are extensions to UML 2.5 use case meta-model are presented in gray in Figure 6.1.

We also propose a new *DirectedRelationship* specialization, named *Refine*. In our proposed meta-model the *Refine* relationship is unidirectional and represents a relationship between a use case and a *UseCaseModel*. A use case can only be refined in one *UseCaseModel* and a *UseCaseModel* can only refine one use case. This new relationship means that the use case being decomposed is related with all the use cases represented in the *UseCaseModel*. A *UseCaseModel* must include, at least, two use cases.

The UML meta-model does not distinguish between abstract use cases and concrete use cases (Metz et al., 2001). In fact, when we need to see the system as whole, the abstract use cases are more useful. But when we need to know the details of the system, the concrete use cases are more useful. The approach presented next starts with abstract use cases and refines them (by decomposing) in order to become concrete use cases. This way we can relate the abstract use cases with corresponding concrete use cases.

The next subsection describes the construction of the decomposition triangle, the name that we have adopted for the method described in next chapter.

6.3 The Decomposition Triangle approach

Azevedo *et al.* propose an extension to the UML meta-model to represent the use case *refine* relationship (Azevedo et al., 2010). As Azevedo *et al.* say, refining use cases means decomposing and simultaneously detailing use cases (Azevedo et al., 2010). This means that when we are decomposing a use case we are adding more details or information about what happens in the system, or how the system should behave.

We are presenting here a top-down decomposition approach based on succes-

sive use case refinement steps. Each use case presented in a use case diagram can, itself, be decomposed in a new use case diagram that refines the use case. The decomposition results in a structure that looks like a triangle, because when we go down in the decomposition, the number of use case diagrams increases.

The decomposition triangle is structured into several levels, starting by a context diagram, and it typically goes down to 3 or 4 levels. The number of levels depends on the system complexity, since, formally, there is no limit.

Next, we are referring to the use case that is to be refined as the “base use case”.

The decomposition process is as follows:

- *Level 0*: The purpose of level zero is to capture the context where the system is going to operate. The system context is defined based on actors and is represented as a context diagram. At this level, the system scope and frontier must be identified as well as the system name. At this level, no use case is identified. The diagram is empty, but all involved actors must be identified and represented. The context diagram description should be a general overview of the system.
- *Level 1*: At the first level, the high-value requirements (that address major functionalities of the system) can be represented. At this level, the first use case diagram is created with the highest abstraction level. At this point, one must think about “which functionalities will satisfy the actor’s needs?”. So, for each actor identified on level 0 it is necessary to discover the main functionalities that the system will provide to satisfy the actor needs. Each functionality is represented as a use case, in the use case diagram, with which relations with the corresponding actor (or actors) are established. The use cases description should be a general overview of the functionalities they address. The system purpose must be well characterized as well as the system scope.
- *Level (i+1)*: Each use case identified in level (*i*) can be detailed in another use case diagram, represented in the next level (*i+1*). The procedure is as follows: for each use case identified on level *i* it is necessary to verify if the use case description is concrete, short and easy to understand. If so, it is not necessary to detail the use case, otherwise the use case should be *refined* and decomposed in a use case diagram. We must assure that there is no information lost during the decomposition process. All information included in the description of the base use case must be included in the description of the use cases that belong to the diagram that decomposes it. The abstraction level decreases from one level (*i*) to another (*i+1*) until the last level is reached. While increasing the detail in the use cases, they become more concrete.

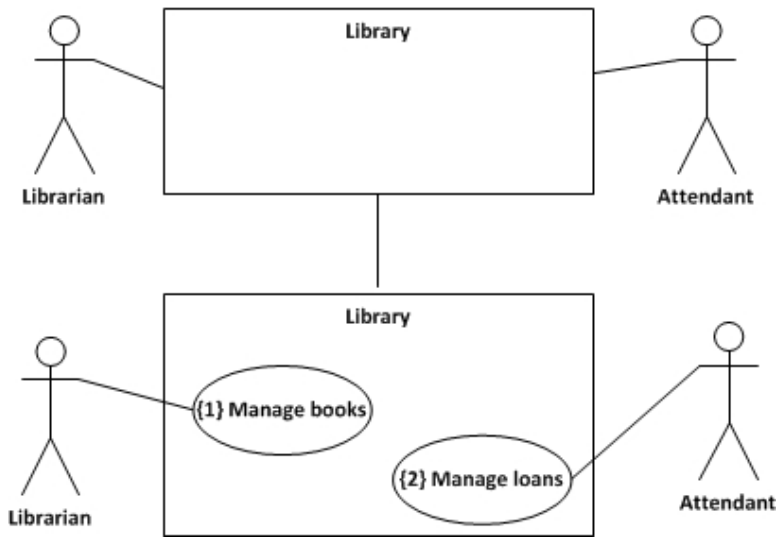


Figure 6.2: The first iteration result

The decomposition is an iterative and incremental process and typically ends when the CRUD (Create, Retrieve, Update and Delete) operations are identified. Each iteration may imply a review of all previous levels.

For better illustrating the proposed approach, we are going to use a running example from a School Library System. In this example when an actor is related with all use cases in a use case diagram, the established association between the actor and the use cases are represented by a single link to the diagram boundary, to simplify the representation. If an actor is associated to only some of the use cases that belong to the use case diagram, the link is established between the actor and the related use cases.

The main purpose of the School Library System is to manage books and loans from a School Library. In the first iteration two actors have been identified: Librarian and Attendant. The Librarian is responsible for managing the books, buy new books, catalog them, check the books' status and so on. The Attendant is responsible for meeting borrowers and for managing the books' loans to the borrowers.

A first proposal for the context diagram and the use case diagram level 1 is shown in figure 6.2. On the first use case diagram (*level 1*) two main use cases are identified: "Manage books" and "Manage loans". The use case descriptions are presented in Table 6.1.

Sometimes the first assessment about "who the actors are" is not correct. Moreover, during the decomposition process the knowledge about the system increases. For instance, new actors can be identified, existing actors may be specialized, new relationships between existing actors and use cases can be found. Occasionally, the very name of the system may be modified to well fitting the

{1} Manage books: The Librarian is responsible for purchasing new books to the library. To do that, the Librarian examines the requests made by Teachers, for the acquisition of new books. The Librarian shall be able to list the books information.

{2} Manage loans: The Attendant is responsible for the books' loans management. Only Students and Teachers can borrow books. A book can only be borrowed by a given number of days. After this number of days the book must be returned. If a book is already lent to someone else, the Borrower could reserve it to himself/herself by appending his/her name to the reservation list. When a book is reserved to someone the delivery date can't be postponed. Otherwise, the Borrower can postpone the delivery date or can ask to the attendant to do that. The Attendant, as well as the Borrower, can list the books available on the library or consult the information about a specific book.

Table 6.1: Use cases descriptions (*level 1*) - first proposal

system purpose, and the use cases have to be redesigned. As a consequence, each process iteration may imply changes to all levels.

In our running example, reading the “{1} Manage books” use case description (Table 6.1) we can clearly identify the Librarian as an actor, but Teachers are also involved in the use case because the Teachers make requests for new books acquisition. This way we may conclude that the Teacher interacts with the system, so, Teacher is also an actor. In the next iteration, the Teacher should be added as an actor in every level.

Every time an actor specialization is identified, the context diagram must be updated because all actors must be represented at the context level, including all actors' hierarchies.

In our running example, analyzing the “{2}Manage loans” use case description in Table 6.1, we can perceive that the Borrower can renew a loan, list information about the books and about the loans, reserve a book, so the Borrower interacts with the system. Consequently, the Borrower should be considered as an actor. Moreover the Borrower can be specialized as a Student or as a Teacher, so we have an actor's hierarchy. This actor's hierarchy must be reflected in the context diagram on the next iteration (see Figure 6.3).

In the first iteration's, *level 1* diagram (Figure 6.2), we can see that the use case names are too general, and based on the use case description we can identify more use cases. Since we only have two use cases in that diagram, it needs to be redesigned. These changes are also reflected in the next iteration (*level 1*) use case diagram (Figure 6.3).

Figure 6.3 represents the result of the second iteration, where the context

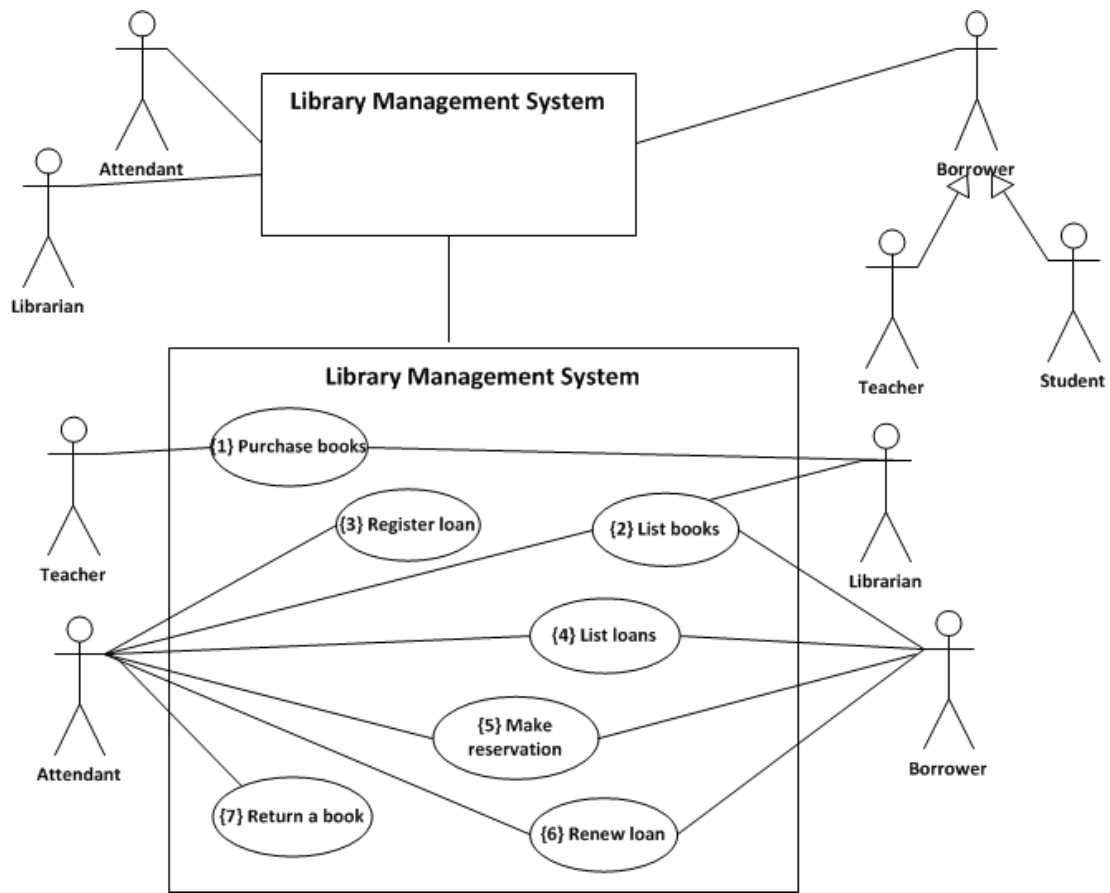


Figure 6.3: The second iteration result

{1} Purchase books: The Librarian is responsible for purchasing new books to the library. To do that, the Librarian examines the requests made by the Teachers, for the acquisition of new books. The Librarian also verifies the financial plan, elaborates the order to buy the books and manages the books' payment. After the books arrival the librarian catalogues them.

{2} List books: The Librarian shall be able to list the books information. The Attendant and the Borrower can check the books available on the library. They can list the books of an author, by book title, by ISBN, etc. They can also consult a book details including a summary about the book's content.

{3} Register loan: The Attendant is responsible for the books' loans management. Only Students and Teachers can borrow books. Only the Attendant can register a loan because she/he must verify the Borrower identification. The Borrower must be registered, otherwise the book cannot be lent. But the Attendant can register a new Borrower if she/he has the identification. Before lending a book it is necessary to verify if the book is available. If the book is available the loan can be registered. If the book has a reservation list the Attendant needs to verify if the borrower is the next on the list and the list must be updated.

{4} List loans: The Attendant can list all loans and can view a specific loan details. The Attendant can check all loans whose delivery date is exceeded and can send a warning (by email) to all Borrowers that have loans out of date. A Borrower can only list her/his own loans.

{5} Make reservation: The Students and Teachers can reserve a book for themselves by appending their name to the book reservation list. Teachers have priority over students.

{6} Renew loan: A Borrower or the Attendant can postpone the delivery date but only if the book does not have a reservation list, otherwise the book must be returned. When the delivery date is exceeded it cannot be postponed.

{7} Return a book: When a Borrower returns a book the delivery date must be checked. If the planed delivery date is exceeded, the Borrower must pay a fine. The fine value depends on the number of days exceeded. The Attendant calculates the fine value, presents a bill and receives the payment. The total payments received must be checked at the end of the day or when the Attendant finishes his/her job and is replaced by other Attendant.

Table 6.2: Use cases description (*level 1*) - second iteration

<p>{1.1} Select books to acquire: Whenever it is necessary to buy a new book, teachers fill out a form with the identification of the book with title, ISBN, author, publisher and edition. Periodically the Librarian examines those forms, verifies if there are duplicated requests, and prioritize the requests in order to select the list of books to acquire.</p>
<p>{1.2} Order books: The librarian selects the suppliers, agrees with them the payment mode, and sends the orders to them. The system shall track the money spent on the acquisition of new books.</p>
<p>{1.3} Catalog books: The librarian shall determine the correct shelf to put the book.</p>
<p>{1.4} Add books: The librarian inserts the books information in the system. The book information must be complete (title, ISBN, author, publisher and edition) including the shelf identification and a summary about the book content.</p>

Table 6.3: “{1} Purchase books” use cases description (*level 2*)

diagram has been updated, as explained above, to include the new identified actors and use cases. Since new use cases were identified, the use case descriptions were also transformed. The new use cases descriptions are represented in Table 6.2.

The evaluation about the need of decomposing is firstly based on the use cases description, but the details to be added can be discussed with the stakeholders because at this software development process’ stage the stakeholders are still involved in the process. The stakeholders that help to identify the *level 1* use cases must have a global view of the system. When the use cases are being decomposed, the involved stakeholders must know deeply the use cases in discussion to correctly identify all necessary details.

Based on the use case descriptions the modeler should verify which use cases need to be *refined*. In our example, the “{5} Make reservation” use case corresponds to a creation operation, so it is a CRUD operation. Moreover, the description of this use case is short, concrete, and easily understood, so, this use case doesn’t need to be further *refined*.

The other use cases descriptions are broad, complex and, in some cases, ambiguous. Such descriptions leave us with doubts to be resolved. So, we have to deepen the knowledge about these use cases, i.e., these use cases should be *refined* in the next level.

The refinement result can be seen in Figure 6.4.

The decomposition process continues with the evaluation of use cases level 2 descriptions.

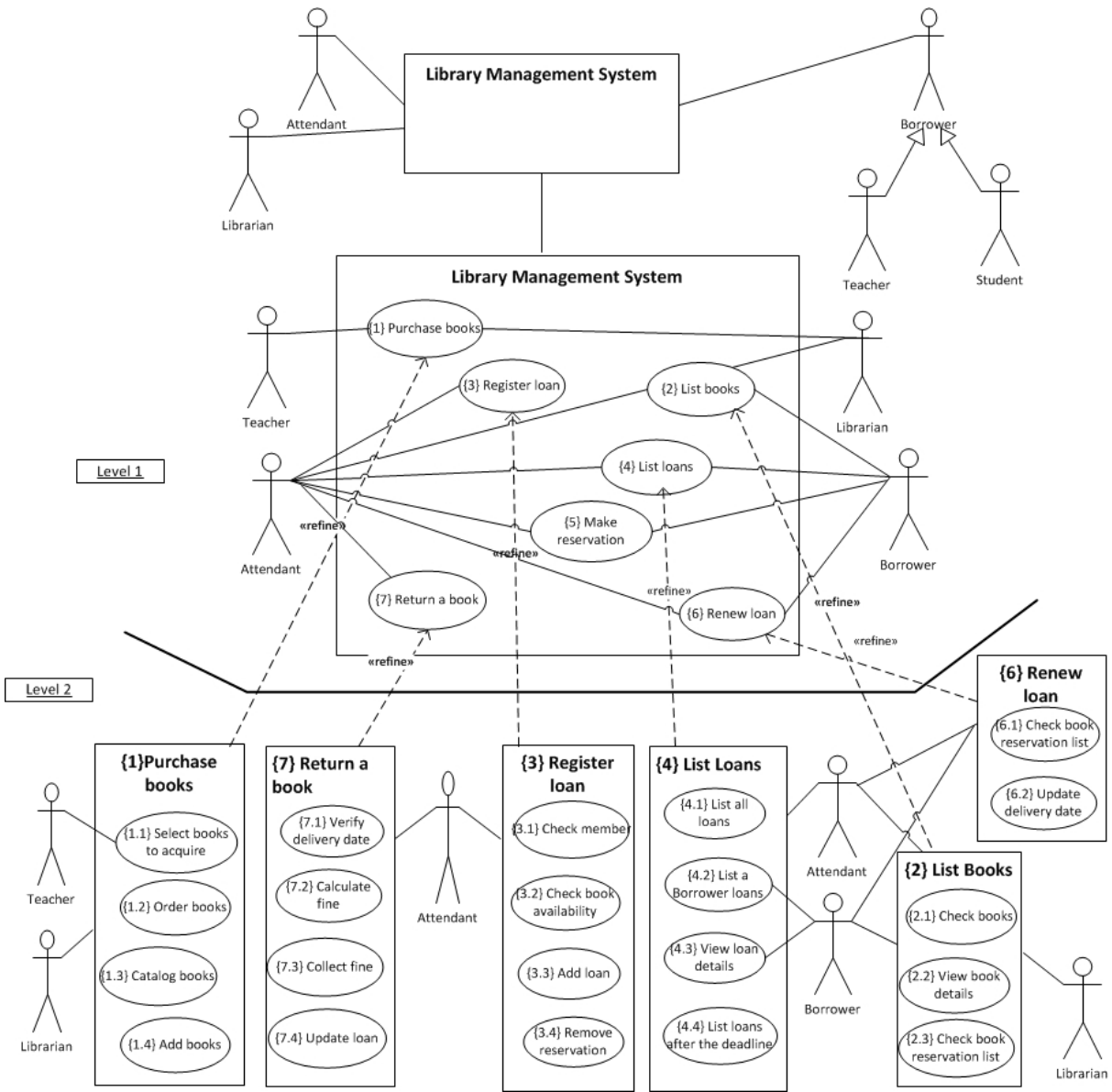


Figure 6.4: The library system decomposition triangle

Use cases like “{1.4} Add books”, “{2.2} View book details”, “{3.3} Add loan”, “{3.4} Remove reservation”, “{4.3} View loan details”, “{6.2} Update delivery date” or “{7.4} Update loan” clearly represent CRUD Operations so they do not need to be refined. The remaining use cases have to be analyzed.

To avoid extending the text too much, we are going to proceed our example by focusing our attention on the “{1} Purchase books” use case. Thus, only the descriptions of the use cases belonging to the “{1} Purchase books” use case diagram are presented in Table 6.3.

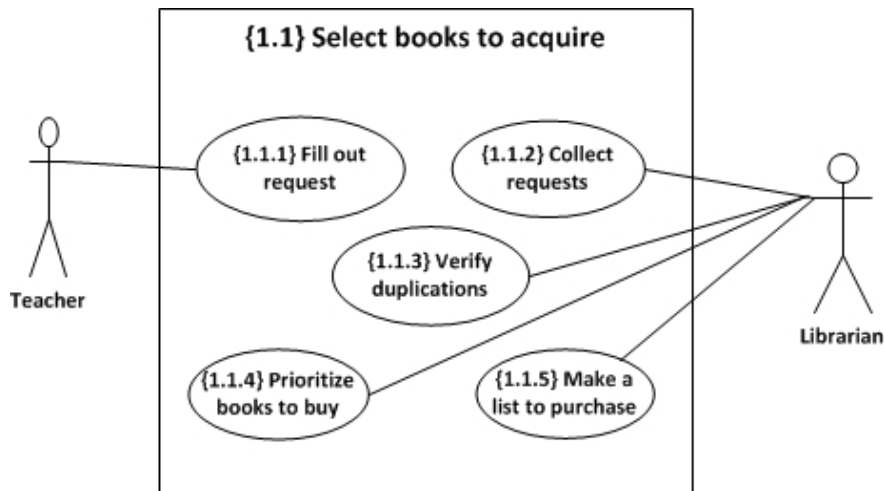


Figure 6.5: “{1.1} Select books to acquire” use case diagram (*level 3*)

The “{1.3} Catalog books” and “{1.4} Add books” use case descriptions are clear and short. The other two use cases need to be *refined*. The refinement of “{1.1} Select books to acquire” use case is represented in Figure 6.5 and the descriptions are presented in Table 6.4.

The refinement process of the branch initiated in the use case “{1} Purchase books” ends because all use cases descriptions are short, concrete and are almost CRUD operations. When all branches decomposition reach such status, the whole decomposition triangle will be complete.

The refinement rules

In this subsection we enumerate and explain the rules used in the proposed approach:

1. The *refine* relationship is unidirectional. The relationship is established between use case diagram represented in the next level ($i+1$), and a base use case, represented in level i , which it refines. The use case diagram has the same name, and number, as the base use case. For example, we can see in Figure 6.4 that the “{3} Register loan” base use case is decomposed,

{1.1.1} Fill out request: Teachers fill out a form with the identification of the book with title, ISBN, author, publisher and edition. The Teacher must be registered on the system. The registration must include the Teacher number, name and department.

{1.1.2} Collect requests: Periodically, the Librarian collects all requests and check for books requested that are already on the Library.

{1.1.3} Verify duplications: The Librarian should check for duplicate requests. For each duplicate request the Librarian should increase the number of request for the book.

{1.1.4} Prioritize books to buy: The Librarian may prioritize the requests by several criteria: by number of request, by request date, by department.

{1.1.5} Make a list to purchase: Depending on the budget the Librarian can order all books or order only the books with highest priority.

Table 6.4: “{1.1} Select books to acquire” use cases description (*level 3*)

and refined, in the “{3} Register loan” use case diagram. This relationship means that the set of use cases, represented in the use case diagram (level $(i+1)$), together decomposes and refines the base use case (level i).

2. When a use case is *refined*, the use case diagram that refines it must have, at least, two use cases, each one with a more detailed description, as we can see in the “School Library System” example.
3. All identified use cases are numbered using the tag=value UML mechanism. The use case number identifies the use case level in the decomposition triangle and vice-versa. For example, the use cases {1}, {2}, {3}, {4}, {5}, {6} and {7} are represented in level 1. The use cases {1.1}, {1.2}, {2.1}, {2.2} are represented in level 2, the use cases {1.1.1}, {1.1.2}, etc. belong to level 3.

The numbering identifies the decomposition chain from the diagram level 1 to each use case. The last numbering level distinguishes the use cases that belong to the same diagram. For example, the use cases {1.1.1} and {1.1.2} belong to the diagram that decomposes the {1.1} use case. The {1.1} use case belongs to the diagram that decomposes the {1} use case.

4. If an actor is related with a base use case (level i), this actor (or a specialization of it) must be related with, at least, one use case in the use case diagram that decomposes it on level $(i+1)$ and vice-versa. As we can see

in our example (Figure 6.4), Teacher and Librarian are related with “{1} Purchase books” base use case (*level 1*), so this two actors are also related with the use cases belonging to “{1} Purchase books” diagram (*level 2*).

5. If a use case is related with only one actor, this actor (or specializations of it) must be related with all use cases in the use case diagram that decomposes it. In our example we can see that the Attendant is the only actor related with “{3} Register loan” use case, so, the Attendant is related with all uses cases in the “{3} Register loan” use case diagram.
6. During the decomposition we have to ensure that every decomposed use case leads to a progress in terms of knowledge about the use case, without loss of information. The information contained in the set of use cases, represented in one use case diagram (*level (i+1)*), together, must exceed the base use case information (*level i*).
7. In the end of the decomposition, all actors must be represented in the context diagram, including all actors’ hierarchies, and all actors must be represented on the first decomposition triangle level (*level 1*), as we can see in Figure 6.4.

The use case decomposition originates a tree structure. The context diagram represents the tree root and serves as an entry point to the navigable structure. The tree root corresponds to the maximum abstraction level. The set of all tree leaves corresponds to the maximum detail we can reach about one possible use case-driven functional description of one system.

The use cases numbering can be used to relate the use cases in a navigable tree structure, helping (this way) to drill down and trace back from a leaf use case to the root and helping to relate and to reintegrate the use case diagrams.

This approach allows us to ensure that no use case (and no actor) is forgotten and no use case (or actor) appears from nowhere. Furthermore this way the uses cases can be easily traced back and the use case diagrams reintegration is possible.

The resulting decomposition triangle can be very useful in many circumstances in the following stages of the software development process. However, in all real cases in which this approach has been used by our research team, the decomposition results were mainly used by the 4SRS method. In fact, the decomposition triangle and the 4SRS method complement each other.

In the next section, we briefly explain how the results from the decomposition triangle can be used within the adoption of the 4SRS method.

6.4 The 4SRS Method and the Decomposition Triangle

One of the most complex activities during software systems development is the transformation of a requirements specification into architectural design models (Fernandes et al., 2000). This is the main purpose of the 4SRS method. The 4SRS method ensures the transition from user requirements specified in use case models into software logical architectures, objects and class models. These models serve as a basis to the software development teams.

The 4SRS is a method organized in 4 steps that transforms use case models into architectural elements (Fernandes et al., 2006; Machado et al., 2006). A complete description of the 4SRS method can be found in (Machado et al., 2005).

This method was created in 2000 (Fernandes et al., 2000) and since then the method has been used in several projects and was extended in several ways. The extension proposed in (Santos and Machado, 2010) appends two more steps to the method in order to generate the class diagram, namely class creation and class characterization.

The decomposition triangle starts with high abstraction level use cases. The abstraction level decreases in every use case decomposition. When the abstraction level decreases the use case description shall be enriched with details that may become useful to the 4SRS purpose which is the generation of architectural models to be used as a basis for software development. When we are refining a use case, all information we have must be represented on the use case diagram that refines it and preferentially with more detail. During the refinement process we do not lose any information.

As we said before the use case decomposition triangle originates a tree structure. If we bring together all tree leaves' use cases, and their descriptions, we will collect the most detailed and complete information we can get.

The resulting decomposition triangle represented in Figure 6.4, assuming that the refinement process ends at this decomposition level, originates the tree structure represented in Figure 6.6.

From the tree structure represented in Figure 6.6, the 4SRS will select and manage all use cases painted in gray (the leaf use cases) and only with the leaves, to avoid working with redundant information, since during the decomposition process all information that we have in a use case (*level i*) must be carried to its refining use case diagram (*level i+1*).

Based on the use case names and descriptions, the 4SRS allows to obtain the class diagrams, as described in (Santos and Machado, 2010) and the architectural elements (Fernandes et al., 2006; Machado et al., 2006). Figure 6.7 illustrates the application of the 4SRS method that transforms leaf use cases, resulting from the decomposition triangle approach, into a logical software architecture.

The 4SRS method executes a series of validations and adjustments to the

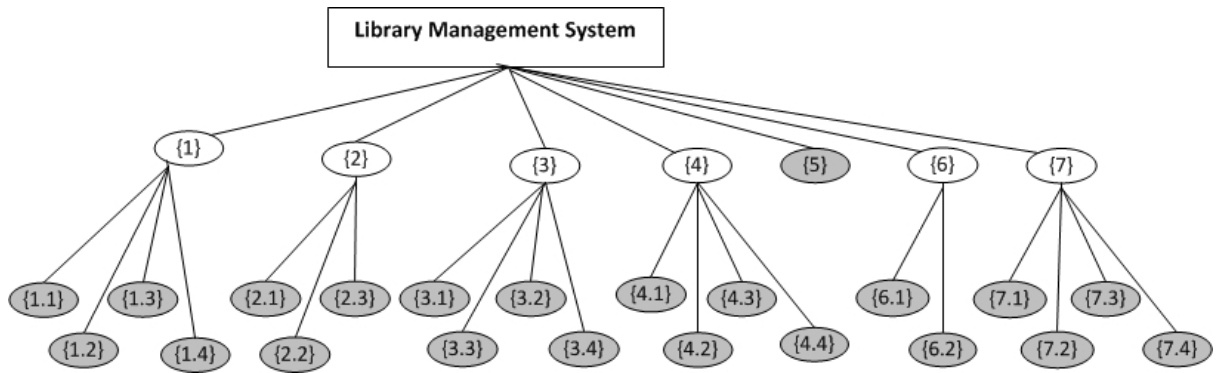


Figure 6.6: A tree structure example

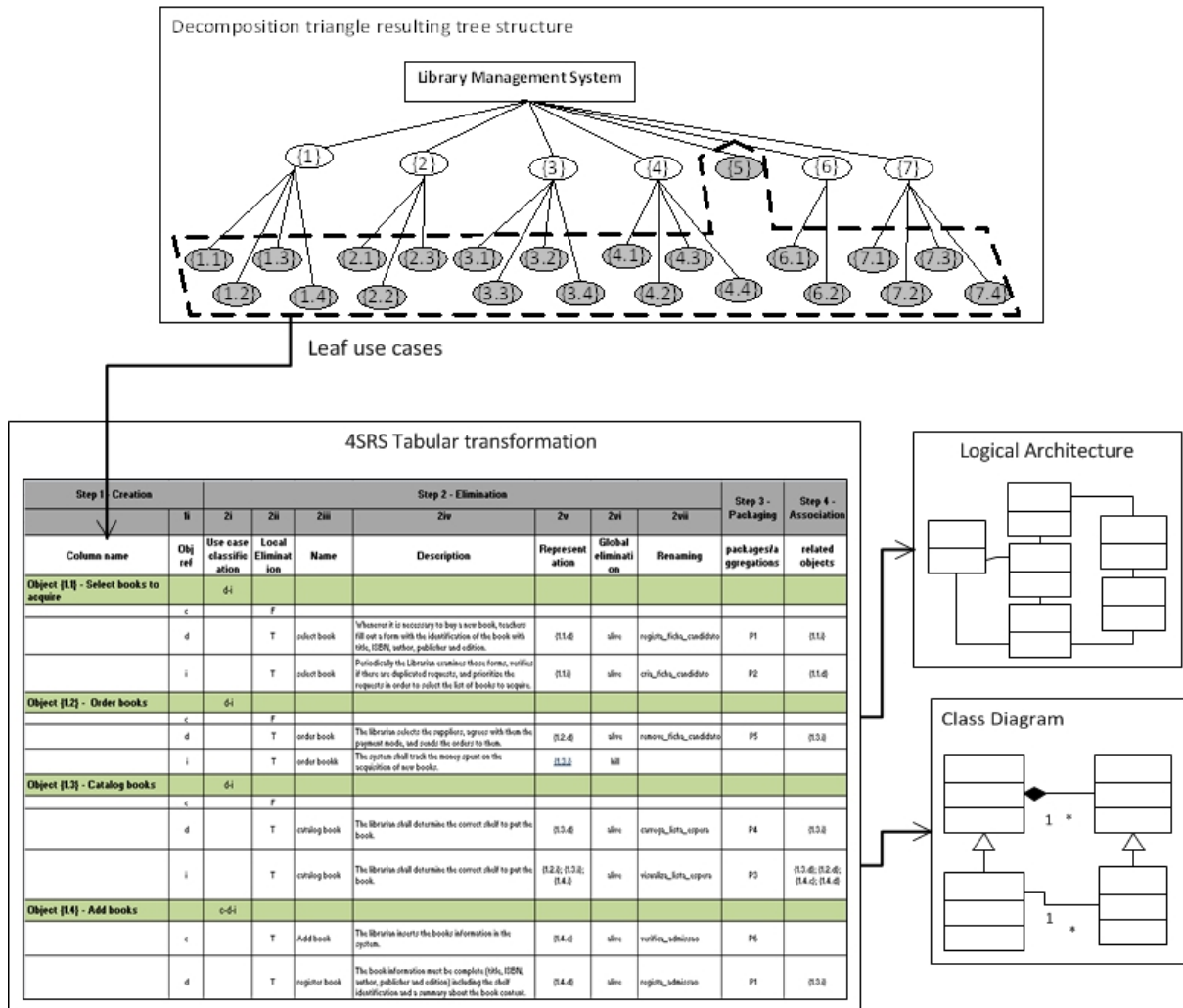


Figure 6.7: The 4RSR transformation

original use case set, so it is important to have a well-established decomposition triangle. This may, sometimes, demand changes to the resulting decomposition triangle before applying the 4SRS, which may imply an additional iteration of the decomposition process or a decomposition review. On the other hand there are some issues like the use cases duplication that are handled by the 4SRS method. In our example we can see (Figure 6.4) that the “{2.3} Check book reservation list” and “{6.1} Check book reservation list” use cases are clearly the same use case. The 4SRS will detect that and deal with it, as explained in (Fernandes et al., 2006; Machado et al., 2006; Machado et al., 2005).

The 4SRS method and the decomposition triangle process are interrelated. The 4SRS method performs a set of validations to verify if the set of use cases resulting from the decomposition process is consistent. If it is not the case it may demand a decomposition process review. On the other hand, the decomposition process does not need to worry about use case reuse or reintegration because the 4SRS will handle it.

The presented approach has already been used and tested in several real and complex projects like (Ferreira et al., 2012; AAL4ALL, 2012). In all that real projects, the decomposition triangle approach and the 4SRS method were used together. First the decomposition triangle approach decomposes the use cases in order to prepare the user requirements to be used by the 4SRS method to generate the logical software architectural design.

By combining the 4SRS with the second approach presented in next section it is possible to generate the software logical architectures and class diagrams based on a set of interrelated business process models allowing tracing back from software models to the business processes models and from the business processes to the corresponding software models.

6.5 Deriving a Use Case Model from a Set of Interrelated Business Process Models

In the approach presented in this section we intend to aggregate and merge the information we have in a set of interrelated business processes into one integrated use case model. To do that we are using a decomposition triangle approach presented in section 6.3.

A business process model identifies the activities, resources and data involved in the creation of a product or service, having lots of useful information for starting to develop a supporting software system. With regard to software development, one of the most difficult and crucial activities is the identification of system functional requirements. A popular way to capture and describe those requirements is through UML use case models. Usually an organization deals with several business processes. As a consequence, a software product does not

usually support only one business process, but rather a set of business processes.

This section presents an approach that allows aggregating in a use case model all the information that can be extracted from the set of business process models that will be supported by the software under development. The generated use case model serves as a basis for the software development process, helping reducing time and efforts spent in requirements elicitation. This approach also helps to ensure the alignment between business and software, and enables traceability between business processes and the corresponding elements in software models.

Cockburn (Cockburn, 2001) distinguishes different use case abstraction levels. The low detail level use cases, and the very detailed level use cases with a clear intention. Cockburn states that low detail level use cases are not trustable as functional requirements for the system being built (Cockburn, 2001).

From another point of view, Cockburn categorizes use cases as business use cases and system use cases (Cockburn, 2001). Cockburn sees business use cases as low detail level (high abstraction) use cases and system uses cases as high detail level (low abstraction) use cases and advises the use case writers to start with the business use cases and “unfold” them continuously until they become system use cases (Cockburn, 2001).

The approach we are presenting herein is aligned with Cockburn’s points of view, helping in transforming business process models (low detail level use cases) into system level functional requirements (high detail level use cases), and helping in keeping track of the transformations of the use cases.

All surveyed existing approaches (section 3.4) obtain a use case diagram based on a single business process model. None of the surveyed approaches aggregates a set of business processes models in one use case model including the use case descriptions¹. But, typically, in a real situation, a software product does not support only one process, but a reasonable set of processes. In order to generate useful use case model it will be necessary to consider the set of business process models that will be supported by the software under development.

The set of business processes, belonging to an organization, being supported by the software under development must be grouped in a single use case model because a software development team needs to understand the system context and scope before starting to plan and design a solution. Following this reason, first it is needed to identify and specify which business processes are to be supported by the software under development.

A use case model can be created with a high abstraction level or low abstraction level (Cockburn, 2001; Cruz et al., 2014c). The approach we are presenting in this section starts with high abstraction level use cases and ends with lower abstraction level use cases. The approach starts by grouping all processes that will be supported by the software under development in one use case diagram

¹The generation of use case descriptions from business process models is highlighted in section 5.2.

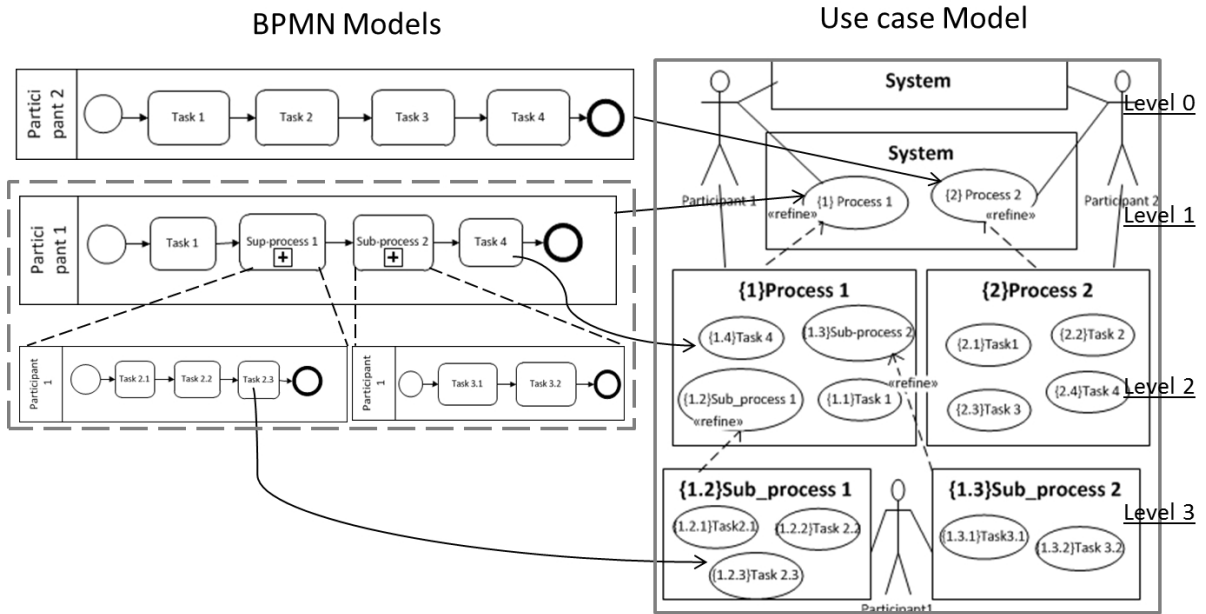


Figure 6.8: A generic decomposition scheme

where each process is represented as a use case. Each use case is then refined and decomposed in a use case model as described in section 6.3. All identified use cases are numbered using the *tag=value* UML mechanism. A generic scheme of applying the decomposition is shown in Figure 6.8.

Similarly to what happens in the decomposition triangle approach, the approach represented in this section the use case models in different abstraction levels:

- **Level 0:** At this level, the system scope and frontier must be identified as well as all the actors involved, so the set of business processes that will be supported by the software under development must be identified. Each participant in a business process model (represented as a pool or a lane) is transformed in an actor (with the same name) in the use case diagram (see Figure 6.8, level 0). The subdivision of a pool in several lanes, or a lane in other lanes originates an actor's hierarchy (Cruz et al., 2014b), as described in section 5.2. Thus, at this level all participants involved in the set of business processes are represented, as actors, in the actor's diagram.

Usually a participant is involved in several business processes belonging to an organization. When participants with the same name are involved in more than one business process we assume that they represent the same participant, and, as consequence, they will be represented by the same actor in the use case model.

- **Level 1:** At this level, the first use case model is created with the highest

abstraction level where each top level business process is transformed into a use case, in the use case model (a business use case) (see Figure 6.8, level 1). Each use case, representing a business process, is related with the corresponding actors representing the participants involved in the process. The use cases are numbered sequentially. The use case description is a general overview of the process it addresses.

Monsalve *et al.* state that a business process model does not allow identifying business process goals or main objectives, but that is important to software requirements elicitation (Monsalve et al., 2012). At this level, the use case description can be used to represent the business process goal and objective helping to understand the business process purpose. That can be accomplished by talking/discussing with stakeholders since at this software development process' stage the stakeholders are still actively involved in the process.

- **Level 2:** At this level, each process (represented as a use case in level 1) is mapped to a use case model following the rules described in section 5.2. Basically one activity from a business process is transformed into a use case and each participant is transformed into an actor. All incoming and outgoing connection flows from the activity originate a NL sentence in the description of the use case that represents the activity. Each generated use case model in *level 2* refines and decomposes a corresponding use case in *level 1* (see Figure 6.8, level 2). The number of use case models, at this level, is the same as the number of use cases existing in the diagram at *level 1*. The use cases identified are numbered using a leveled numbering.
- **Level ($i+1$), ($i \geq 2$):** At this level, each use case that represents a sub-process in *level i* is decomposed and refined in a use case model in *level ($i+1$)* (see Figure 6.8, level 3). BPMN has two types of activities: a task (atomic activity) and a sub-process (OMG, 2011a). A sub-process is a process consequently the use case that represents the activity can be decomposed and mapped to a use case model. The decomposition ends when all use cases representing processes or sub-processes are decomposed and refined.

In the presented approach, refining a use case means detailing all activities involved in the corresponding process, including all resources and/or data that are consumed and produced, messages exchanged, decisions that have to be taken, events that can occur, etc.

The decomposition results in a tree structure where the leaf nodes represent the tasks and the non-leaf nodes represent the processes and sub-processes. The decomposition tree has high abstraction level use cases at *level 1*. The abstraction level decreases in every use case decomposition. The approach allows to relate use cases belonging to different abstraction levels allowing to drill down and up

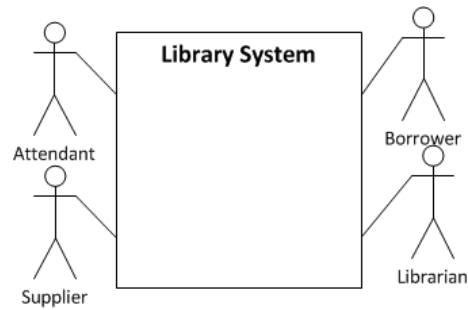


Figure 6.9: Actors diagram (level 0)

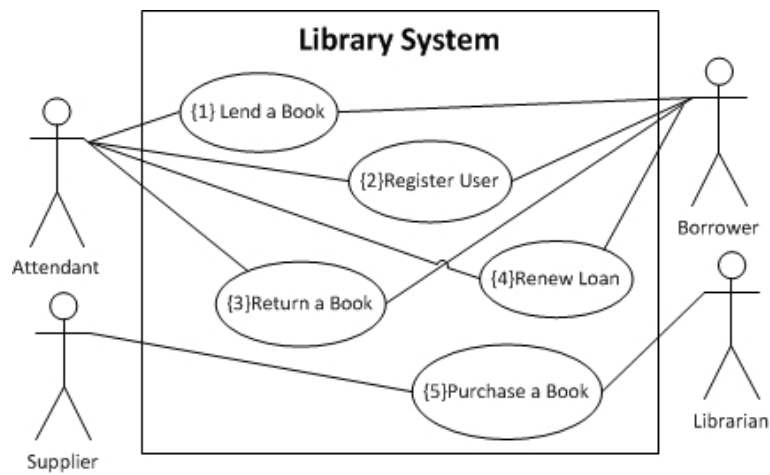


Figure 6.10: Use Case diagram (level 1)

between different abstraction levels. This way, the approach allows tracing back from requirements to the business processes and from the business processes to the corresponding requirements.

6.6 Demonstration Case Aggregating a Set of Business Process Models

In this section we use, as a demonstration case, the example of a School Library System where some of the business process more commonly used have been selected to present here: *Register User*, *Lend a Book*, *Purchase a Book*, *Return a Book* and *Renew Loan*. The *Return a Book* business process model includes the sub-process, *Penalty treatment*.

Figure 6.9 shows the use case diagram level 0 (actors diagram) with four actors, which were derived from the business process participants. The actors are also depicted in diagram level 1 as seen in Figure 6.10.

In the *Purchase a Book* business process model (represented in Figure 6.11) the

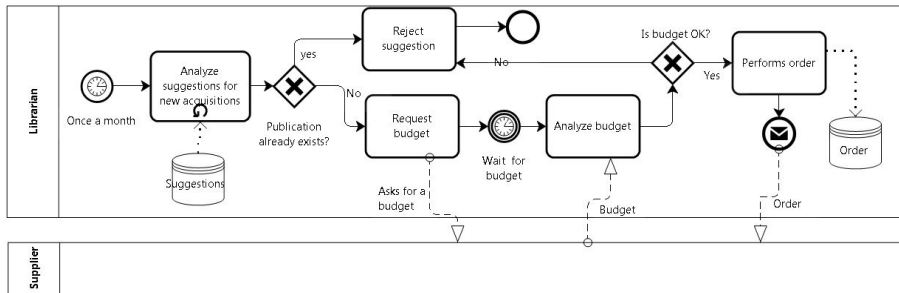


Figure 6.11: Purchase a Book business process model

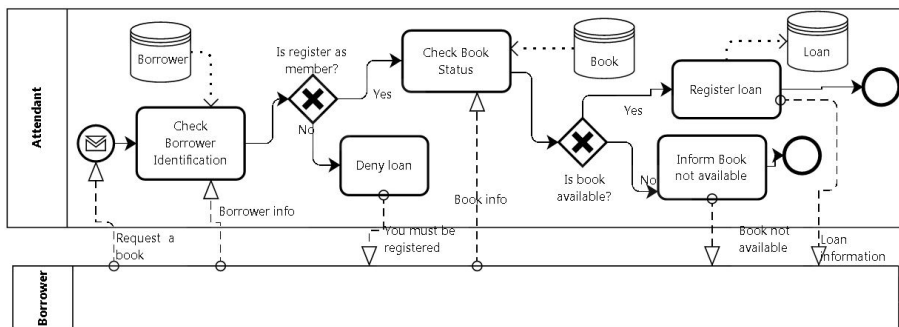


Figure 6.12: Lend a Book business process model

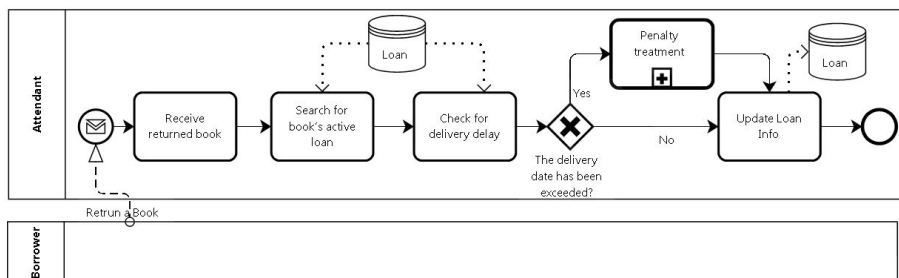


Figure 6.13: Return a Book business process model

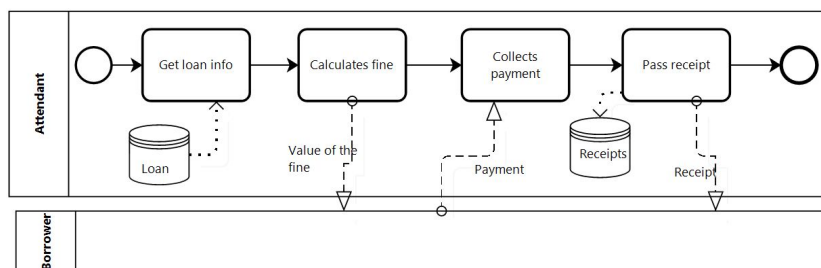


Figure 6.14: Penalty Treatment business process model

participants involved are *Librarian* and *Supplier*. Following the rules presented previously, *Librarian* and *Supplier* must be represented as actors in the actors diagram *level 0* and in use case diagram *level 1* (see Figure 6.10).

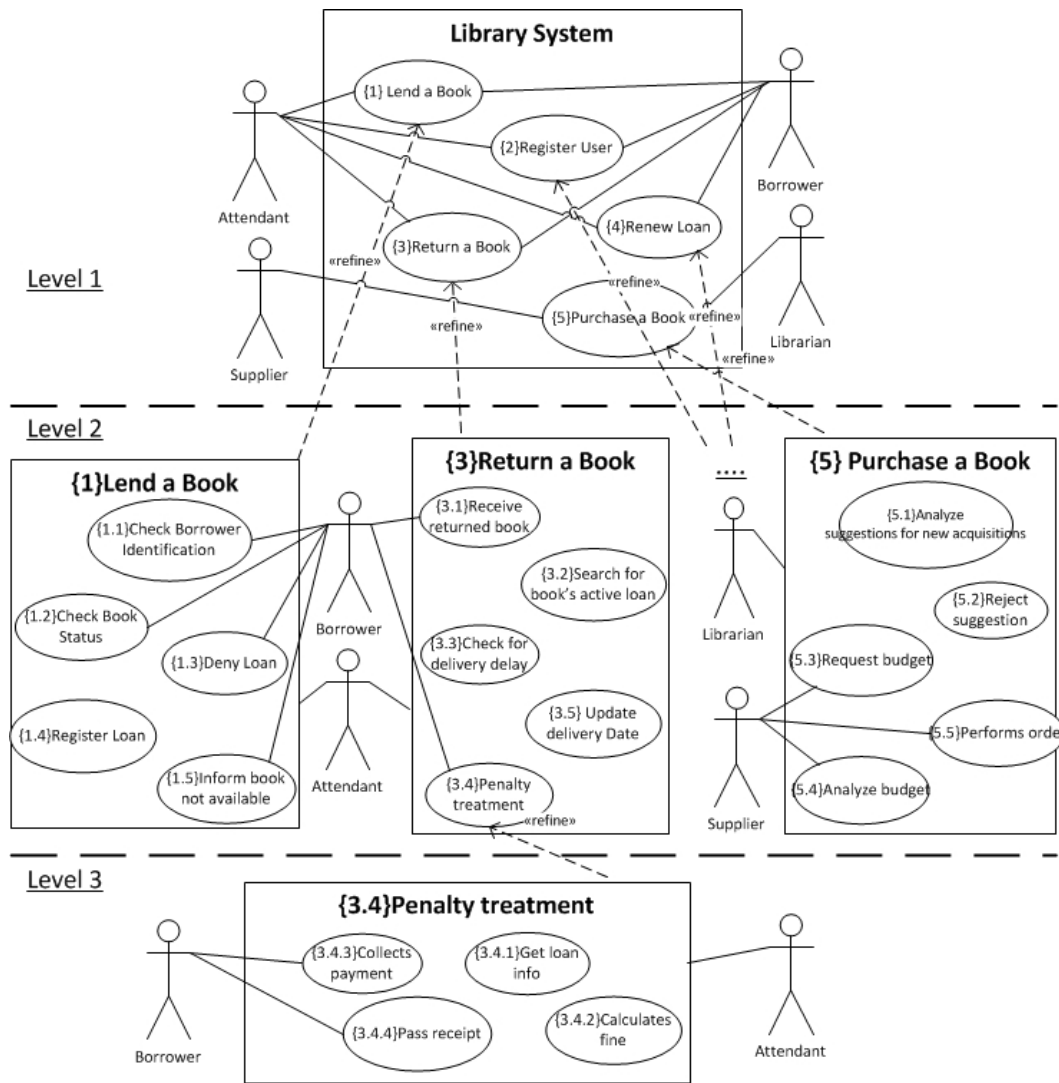


Figure 6.15: Complete Use Case model (level 3)

In *Lend a Book* business process model (Figure 6.12) two participants are identified, *Borrower* and *Attendant*. The two corresponding actors (with the same name) will be also represented in the actors diagram and in the use case diagram *level 1* (see Figure 6.10).

In *Return a Book* business process (Figure 6.13), the participants involved are *Borrower* and *Attendant* already identified in *Lend a Book* business process model. So, the actors belonging to the actors diagram and use case diagram *level 1* remain the same.

The *Register User* and *Renew Loan* business process models are not presented here, but in both processes the participants involved are the same *Borrower* and *Attendant*. The corresponding actors are already represented in the actors diagram and in the use case diagram level 1.

Summarizing, and looking at the business processes we have selected for our demonstration case, we can see that *Librarian* and *Supplier* are the participants involved in *Purchase a Book* business process, *Borrower* and *Attendant* are the participants involved in all the other processes. Thus, these four participants are represented as actors in the actors diagram (level 0), as seen in Figure 6.9.

In the use case diagram level 1 (refer to Figure 6.10), each selected business process is represented as a use case connected with the actors that represent the corresponding participants. So, *Purchase a Book* business process is related to actors *Librarian* and *Supplier*. *Lend a Book*, *Return a Book*, *Renew Loan* and *Register User* use cases are related with the actors *Borrower* and *Attendant*. The use cases are numbered sequentially. Each use case, which represents a process, is then detailed in a use case model at level 2, as we can see in Figure 6.15.

By analyzing the *Return a Book* business process model (refer to Figure 6.13), we can see that the process comprises five activities: *Receive returned book*, *Search for book's active loan*, *Check for delivery delay*, *Update Loan Info* and *Penalty treatment*, each one giving origin to a use case in the use case model that refines the *Return a Book* use case (level 2). All activities belong to the *Attendant* pool, so all use cases representing the activities are related with the *Attendant* actor. *Receive returned book* activity receives a message from the *Borrower* participant and *Penalty treatment* activity also exchange messages with the *Borrower* participant, so the *Borrower* actor is related with both corresponding use cases. Similar line of reasoning can be applied to explain the *Lend a Book* (refer to Figure 6.12) and *Purchase a Book* (Figure 6.11) business process models transformation.

The *Penalty Treatment* use case represents a sub-process (represented in Figure 6.14), so it can be represented as a use case model in the next level (level 3), detailing and refining the use case (level 2) as we can see in Figure 6.15.

6.7 Final Remarks

This chapter presented two approaches, the decomposition triangle approach and an approach that, using the decomposition triangle, aggregates in one use case model all the information available in a set of business process models.

The decomposition triangle approach is a top-down use case decomposition that starts with high abstraction level use cases and ends with very detailed level and concrete use cases. The approach is iterative and incremental and starts by identifying the system scope and actors. In the second iteration, the main functionalities the system shall provide are identified and represented as use cases. In this context, we propose an extension to the UML 2.5 use case meta-model

in order to support the refinement of one use case into a use case model. Based on the use case description and, if necessary, by obtaining stakeholders feedback, each use case can be decomposed in a use case model represented on the next decomposition level, decreasing this way the abstraction level and increasing the specification detail of the use cases at that level.

In the second approach presented in this chapter we conclude that using the decomposition triangle approach it is possible to systematically derive a complete use case model from a set of interrelated business process models, helping to ensure that the software requirements really meet business needs. This decomposition process generates a use case tree structure which is especially prepared to feed the tabular transformation table of the 4SRS method, allowing the generation of the logical software architecture and other software models based in business process models and allowing traceability between business processes and use cases.

This second approach starts by identifying the set of business processes that will be supported by the software under development, identifying the system scope. Then a use case model, divided in several abstraction levels, is created based on the set of identified business process models. In level 0, all actors and all actors' hierarchies are identified, representing the involved participants. In level 1, each use case represents a business process being supported by the software system under development. Each use case is then decomposed and refined in a use case model (level 2). The approach ends when all use cases representing processes and sub-processes are decomposed into atomic activities, each one being represented as a use case.

Using the decompositions triangle we can easily “zoom in” to a use case to see the details and “zoom out” to understand the context where the use case appears. Or, by other words, we can easily decrease the abstraction level entering into concrete details or increase the abstraction level when we need to understand the system as a whole.

The decomposition triangle also allows the division of the requirements elicitation task in parts because the reintegration of the parts becomes easy and secure.

Both approaches presented in this chapter are especially useful in a real industrial scenario which complexity and dimension will benefit from a systematic approach for generating the use case model and other software models.

Chapter 7

Conclusion and Future Work

The constant change and rising complexity of organizations, mainly due to the transforming nature of their business processes, has driven the increase of interest in BPM by organizations, leading to the growth of BPM's relevance and importance (van der Aa et al., 2015; Redlich et al., 2014). This increase has been accompanied by the increasing number of theories, modeling processes, modeling languages and notations, and software applications that support BPM.

It is recognized that knowing business processes can help to ensure that the software under development will really meet business needs. Some of software development processes (like Unified Process) already refer to business process modeling as a prerequisite to the next steps of software development. Although most of them do not clarify how the business processes can be used, they see the potential and recognize the advantages of using business modeling for software development (Štolfa and Vondrák, 2008). A business process model is a proper representation of the reality, having several useful information that can be used in the development of the software that will support the business.

The research work presented here uses the information existing in business process models to derive a data model and other software models that can be useful to identify the data model, mainly in contexts of high complexity systems through the use of the 4SRS method. With the goal of obtaining a data model, two approximations have been proposed. Both yield the data model based on a set of business process models, but one of them also obtains other software models, as intermediate models for obtaining the data model, including a use case model and a logical software architecture model.

This chapter discusses and summarizes the obtained results, presents conclusions and points out future research work.

7.1 Overview of the Undertaken Work

The BPMN most recent version has consolidated its importance as a business process modeling language and it is becoming more complete allowing distinguishing persistent from non-persistent data (Aagesen and Krogstie, 2015). This made possible the derivation of data models from business process models, which is the

main aim of this PhD research work. As already mentioned, this was achieved following two different approaches.

A first approach to obtain a data model directly from one business process model is presented in section 4.2. Nevertheless, the presented approach was not able to identify all relationships between the identified entities in the data model, mostly because, in a real situation, a software product does not typically support only one business process, but rather a set of business processes. To generate complete and useful data models for the development of software that will support the business, it is necessary to aggregate the set of processes that comprise a business. Thus, this approach has been extended and improved as presented in the section 4.4. This enhancement aggregates, in one complete data model, the data involved in a set of business processes models.

Because some business process models have bad quality, and in order to deal with high complex systems, another approach to generate a data model based on business process models has been proposed based on the 4SRS method. The 4SRS method has been adapted to deal with a use case model derived from a set of business process models and extended to generate a data model supporting the derived logical architecture (section 5.4). When necessary the generated use case model may be complemented with information from other sources allowing deriving a more complete data model. At this software development stage, the stakeholders are still involved in the process so they can provide useful information to complement the information provided by business process models.

To use the 4SRS, a use case model is needed. Thus, an approach to generate a use case model from a business process model is presented in section 5.2. The approach is focused on obtaining the descriptions of the use cases, approach that uses a set of sentences defined in NL.

A use case model can be created with high abstraction level or with a low abstraction level. A use case model with a very detailed level can be very complex, hard to understand and to manage. In that sense, a process to decompose and refine use cases has been created in order to organize the use cases. The resulting process has been named as “the decomposition triangle”, and is presented in section 6.3. The presented process is very useful to any software development team in order to manage use cases, because it allows the coexistence of both high and low abstraction level use cases, and it enables to trace back and forth between use cases at different abstraction levels.

The decomposition triangle process has been specially designed to aggregate in one use case model, the use cases derived from a set of business processes, which was presented in section 6.5. The process starts by high abstraction level use cases (each use case representing a business process) and ends with low abstraction level use cases (each use case representing an atomic activity). The resulting use case model is especially useful to the 4SRS method, where it can be used to feed the tabular transformation.

Summarizing up, two approaches to derive a data model from business process

models have been proposed in the research work presented herein:

1. A direct approach - by deriving a data model directly from the existing information in business process models.
2. An indirect approach - by using the 4SRS method, which has been adapted to work with a use case model derived from a set of business process models and extended to derive a data model.

The two approaches are able to generate a complete data model including entities, entity attributes and the relationships between those entities, including cardinality and optionality. Still some differences may be pointed out between the two approaches. In Table 7.1 we are comparing the two approaches according to the following aspects:

- Generated models - set of generated software models.
- Degree of automation - level of automation of the created approaches.
- Completeness - level of completeness of the generated software models.
- Correctness - level of correctness of the generated software models.
- Consistency - level of consistency between the derived software models.
- Traceability - level of traceability between the derived software models and the business processes.
- Difficulty - level of difficulty of applying the approach.
- BPMN dependency - level of dependency between the generated software models and the business process models, modeled in BPMN.

Analyzing the table we may see that the direct approach is automatable, demands less work but is totally dependent on the existing information in the BPMN models because the BPMN models are the only source of information. As a consequence, we may say that the direct approach is able to generate correct and complete data models if the BPMN models are complete and correct. Otherwise, the correctness and completeness of the data model are not assured.

The 4SRS extended approach is only partially automatable mostly because the derived software models need to be analyzed and validated by a software engineer and the existent information on BPMN models may be complemented with information from other sources. The 4SRS extended approach generates not only the data model but also the use case model and the logical software architecture based on business process models, which can be used to the development of the software that will support the business.

Aspect	Directly	Using the 4SRS method
Generated software models	Data model.	Use case model (including use case descriptions), logical software architecture and data model.
Degree of automation	Fully automatable.	Partially automatable. The generation of the use case model from business process models is prepared to be automatic as well as the construction of the 4SRS tabular transformation and its first step. The other steps need a software engineer intervention.
Completeness	It can be complete, but relies on existing information in business processes models.	The software models can be more complete since the information derived from BPMN models may be complemented with information provided by other sources.
Correctness	It can be correct, but relies on existing information in business processes models.	It can be more correct because the information can be analyzed and validated by the software engineer during the 4SRS execution.
Consistency	Only the data model is derived.	The use case model, logical software architecture and the data model are consistent between them.
Traceability	Direct.	Direct from BPMN models to corresponding elements in software models. Less direct in the opposite direction because software models may have added information provided by other sources.
Difficulty	Low.	High. The approach involves the participation of a software engineer.
BPMN dependency	Totally dependent. The BPMN models are the only source of information.	Very dependent. All process is based on the information available in the BPMN models. However, the information can be complemented with information provided by other sources.

Table 7.1: Comparing the two presented approaches to derive a data model from business process models

Both approaches allow traceability from elements in software models to the business processes and from the business processes to the corresponding elements in software models. Nevertheless, using the 4SRS, the traceability between the software models and business process models may not be direct because of the information that can be provided by other sources.

7.2 Results and Contributions to the State of the Art

This research work derives a data model and a set of other software models, serving as the basis for software development starting from business process modeling using BPMN. The proposed approaches have been validated through demonstration cases. The research results have been published and discussed in six conference papers and in one doctoral symposium. The published papers are enumerated next:

1. E. F. Cruz, R. J. Machado, and M. Y. Santos, "From business process models to data model: A systematic approach," in 8th International Conference on the Quality of Information and Communications Technology (QUATIC2012), pp. 205-210, IEEE Compute Society, September 2012.
2. E. F. Cruz, R. J. Machado, and M. Y. Santos, "From business process models to use case models: A systematic approach," in Advances in Enterprise Engineering VIII (D. Aveiro, J. Tribolet, and D. Gouveia, eds.), vol. 174 of Lecture Notes in Business Information Processing, pp. 167-181, Springer International Publishing, May 2014.
3. E. F. Cruz, R. J. Machado, and M. Y. Santos, "On the decomposition of use cases for the refinement of software requirements," in 14th International Conference on Computational Science and Its Applications (ICCSA), pp. 237-240, IEEE Compute Society, June 2014.
4. E. F. Cruz, R. J. Machado, and M. Y. Santos, "Derivation of data-driven software models from business process representations," in 9th International Conference on the Quality of Information and Communications Technology (QUATIC2014), pp. 276-281, IEEE Compute Society, September 2014.
5. E. F. Cruz, R. J. Machado, and M. Y. Santos, "Bridging the gap between a set of interrelated business process models and software models," in 17th International Conference on Enterprise Information Systems, pp. 338-345, April 2015.

6. E. F. Cruz, R. J. Machado, and M. Y. Santos, “Deriving a Data Model from a Set of Interrelated Business Process Models,” in 17th International Conference on Enterprise Information Systems, pp. 49-59, April 2015.
7. E. F. Cruz, R. J. Machado, and M. Y. Santos, “Deriving Software Design Models from a Set of Business Processes,” in 4th International Conference on Model-Driven Engineering and Software Development, pp. 489-496, February 2016.

As a conclusion we may say that the objectives of this research work have been accomplished and exceeded. Two ways to derive a data model from business process models have been developed. A direct approach to generate a data model based on a set of business process models, that not only derives a data model but also allows verifying the completeness of the involved business processes (in terms of persistent data) and/or to identify possible links with other applications.

In what concerns to persistent data, interrelated business processes usually complete each other, meaning that the information written by one process is, most of the times, used in another (or the same) business process. Thus, working with a set of interrelated business process models is crucial to be able to derive much more complete software models than when working with only one business process.

However, it is worth noting that if the business process models will provide the basis for the software development, they have to contain relevant information for the development of this software, including the information about the data involved.

Some studies reveal that, usually, business process models have bad quality (Weber et al., 2011). To overcome this issue, a second approach to derive a data model from business process models has been presented, which adapts the 4SRS method to work with the use case model derived from business process models, and extends it to generate a data model. This way the 4SRS allows deriving a logical architecture and a data model supporting the selected set of business process models.

Hence, the resulting contributions to the state-of-art are:

- When the goal is to obtain a data model from a set of complete BPMN models, i.e. containing all information about persistent data, or when the goal is to obtain a data model even if not a complete one, the direct approach presented in chapter 4 is the most useful one. In fact, the direct approach is automatable, and allows deriving a data model from the modeled information available in a set of business process models.
- When the goal is to obtain a data model from a set of BPMN models, which were built at a high abstraction level (business level), neglecting information about persistent data, or when the software that will support

the business is of complex nature, because of the complexity of the business itself or because of the high number of data sets or business roles involved, the approach involving the 4SRS is more suitable, and yields better results. Indeed, this approach derives an intermediate use case model from the set of business process models ((sub)approaches presented in chapter 6), which are used as input to the 4SRS method. The iterative nature of the 4SRS method allows to complete the information about use cases, which were initially derived from the business process models, with information from other sources. This enables completing information that was missing in the initial business process models, and allows obtaining a logical software architecture model, which may then be used to obtain a data model. This demanded an extension to the 4SRS method, presented in chapter 5.

The 4SRS needs the intervention of an expert to analyze and validate the information provided by the set of BPMN models and the derived software models. If necessary, the information provided by the BPMN models may be complemented with information provided by other sources. Thus, the derived software models may be more complete and correct than the software models derived directly from a set of BPMN models, but, at the same time, it can involve much more work and it can be more time-consuming in implementing.

In this case, the generation of a use case model, including use cases descriptions, based on a set of business process models ensures the implementation of all requirements that come directly from the process models. The generated use case model may be used to feed the 4SRS tabular transformation but also can be very useful to a software development team.

In this research we use the DSR research method as is being considered the most suitable because useful artifacts (approaches) were developed helping in closing the gap between the business process modeling and software development. The DSR method focuses on the development of artifacts that extend the current IT limits to drive an improvement of the businesses and organizations performance. In the research presented in this dissertation, two useful approaches have been created, enabling a reduction of the time spent on analysis and modeling software for business support, capitalizing resources, and reducing the risk of creating software products that do not correspond to the real business needs.

7.3 Future Work

As we previously stated, the generation of the use case, including use case descriptions, and the derivation of the data model directly from a set of business process models can be automatized. In the future, it is intended to create a prototype to generate a use case model and the supporting data model based on the same set of business process models. As future work, we intend to apply this research in a real industrial scenario, which complexity and dimension will

benefit from a systematic approach to the identification of the use case model and the supporting data model.

Another future research direction is to extract the data model for decision support systems based on the business process models. These data models involve information for assessing and analyzing the processes, measuring the processes' efficiency and possible delays, allowing to identify bottlenecks and possible needs of business processes reengineering. In that sense, the information for decision support systems is different, although related, from the information for the systems that support the business processes.

When we are designing a logical architecture of an application, we can divide the application into several layers. The communication between those layers needs to be well defined and documented, enabling a good communication between the teams responsible for each layer's development. In every application, internal communication between the distinct architectural layers may be identified, and also external communication between the application being built and other applications.

In BPMN process models, an external participant may represent an external application or someone using the application, such as a customer or a supplier with access to the application, providing or requesting a service, which often may trigger a process execution. Usually, the software that supports the business processes has to interact with applications supporting business partners' processes. In that sense, a software development team needs to know what information is received from, and what information must be sent to external software applications. BPMN language allows identifying the messages and data exchanged between a business process and an external participant. BPMN allows identifying what is received by the business process, from an external participant, and what is sent to the external participant. The identification of the data received by a business process from an external participant and the data sent by the business process to an external participant enables the derivation of the services APIs (Application Program Interface) for interaction between the organizational system, whose processes are modeled by the private BPMN models, and external systems, that support the public BPMN models. As future work we also intend to create an approach to generate these services APIs from the business process models, consistent with the generated data model and with the derived logical architecture.

References

- Aagesen, G. and Krogstie, J. (2015). BPMN 2.0 for modeling business processes. In vom Brocke, J. and Rosemann, M., editors, *Handbook on Business Process Management 1*, International Handbooks on Information Systems, pages 219–250. Springer Berlin Heidelberg.
- AAL4ALL (2012). <http://www.aal4all.org>.
- Abi-Antoun, M., Wang, D., and Torr, P. (2007). Checking threat modeling data flow diagrams for implementation conformance and security. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 393–396, New York, NY, USA. ACM.
- Aguilar-Savén, R. S. (2004). Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129 – 149. Production Planning and Control.
- Allweyer, T. (2010). *BPMN 2.0 - Introduction to the standard for business process Modeling*. Books on Demand GmbH, Norderstedt.
- Alter, S. (2015). How should business informatics integrate service, process, work system, and enterprise orientations? *IEEE Conference on Business Informatics*, 1:1–8.
- Andersson, B., Bergholtz, M., Edirisuriya, A., Ilayperuma, T., and Johannesson, P. (2005). A declarative foundation of process models. In Pastor, O. and Falcão e Cunha, J., editors, *Advanced Information Systems Engineering*, volume 3520 of *Lecture Notes in Computer Science*, pages 233–247. Springer Berlin Heidelberg.
- Avison, D., Lau, F., Myers, M., and Nielsen, P. A. (1999). Action research. *Communications of the ACM.*, 42.
- Azevedo, S., Machado, R. J., Braganca, A., and Ribeiro, H. (2010). The UML «include» relationship and the functional refinement of use cases. In *36th EUROMICRO Conference on Software Engineering and Advanced Applications*.

- Barjis, J. (2008). The importance of business process modeling in software systems design. *Science of Computer Programming*, 71(1):73 – 87.
- Barquet, A. P. B., Cunha, V. P., Oliveira, M. G., and Rozenfeld, H. (2011). Business model elements for product-service system. In *Functional Thinking for Value Creation*, pages 332–337. Springer.
- Batoulis, K., Meyer, A., Bazhenova, E., Decker, G., and Weske, M. (2015). Extracting decision logic from process models. In Zdravkovic, J., Kirikova, M., and Johannesson, P., editors, *Advanced Information Systems Engineering*, volume 9097 of *Lecture Notes in Computer Science*, pages 349–366. Springer International Publishing.
- Baumann, F., Hussein, R. E., and Roller, D. (2015). State of the art of bpm - approach to business process models and its perspective. *International Journal of Electronics Communication and Computer Engineering*, 6:649–457.
- Becker, J., Rosemann, M., and von Uthmann, C. (2000). Guidelines of business process modeling. In van der Aalst, W., Desel, J., and Oberweis, A., editors, *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 241–262. Springer Berlin Heidelberg.
- Bera, P. and Evermann, J. (2014). Guidelines for using uml association classes and their effect on domain understanding in requirements engineering. *Requirements Engineering*, 19(1):63–80.
- Berardi, D., Calvanese, D., and Giacomo, G. D. (2005). Reasoning on UML class diagrams. *Artificial Intelligence*, 168:70 – 118.
- Berenbach, B. (2004). The evaluation of large, complex UML analysis and design models. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 232 – 241.
- Bichier, M. and Lin, K. J. (2006). Service-oriented computing. *Computer*, 39(3):99–101.
- Bittner, K. and Spence, I. (2003a). *Applying use cases: a practical guide*. Pearson Education Inc.
- Bittner, K. and Spence, I. (2003b). *Use Case Modeling*. Pearson Education Inc.
- Bolloju, N. and Leung, F. S. K. (2006). Assisting novice analysts in developing quality conceptual models with UML. *Communications of the ACM*, 49(7):108–112.

- Booch, G., Rumbaugh, J., and Jacobson, I. (1998). *The Unified Modeling Language User Guide*. Addison Wesley.
- Borger, E. (2011). Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns YAWL. In *Software and Systems Modeling - Springer*.
- Bork, D. and Fill, H.-G. (2014). Formal aspects of enterprise modeling methods: A comparison framework. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 3400–3409. IEEE.
- Bragança, A. and Machado, R. J. (2006). Extending UML 2.0 metamodel for complementary usages of the «extend» relationship within use case variability specification. In *10th International Software Product Line Conference (SPLC'06)*.
- Brambilla, M., Preciado, J. C., Linaje, M., and Sanchez-Figueroa, F. (2008). Business process-based conceptual design of rich internet applications. *Web Engineering, International Conference on*, 0:155–161.
- Braun, R. and Esswein, W. (2014). Classification of domain-specific BPMN extensions. In Frank, U., Loucopoulos, P., Pastor, O., and Petrounias, I., editors, *The Practice of Enterprise Modeling*, volume 197 of *Lecture Notes in Business Information Processing*, pages 42–57. Springer Berlin Heidelberg.
- Brdjanin, D. and Maric, S. (2013). Model-driven techniques for data model synthesis. *Electronics*, 17(2):130–136.
- Brdjanin, D., Maric, S., and Gunjic, D. (2011). Adbdesign: An approach to automated initial conceptual database design based on business activity diagrams. In *Advances in Databases and Information Systems*, pages 117–131. Springer.
- Buscemi, M. and Sassone, V. (2001). High-level petri nets as type theories in the join calculus. In Honsell, F. and Miculan, M., editors, *Foundations of Software Science and Computation Structures*, volume 2030 of *Lecture Notes in Computer Science*, pages 104–120. Springer Berlin Heidelberg.
- Bézivin, J. (2006). Model driven engineering: An emerging technical space. In *Generative and Transformational Techniques in Software Engineering*, volume 4143 of *Lecture Notes in Computer Science*, pages 36–64. Springer Berlin Heidelberg.
- Cardoso, E. C. S., Almeida, J. P. A., and Guizzardi, G. (2009). Requirements engineering based on business process models: A case study. In *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th*, pages 320–327. IEEE.

- Carvalho, J. A. (2012). Validation criteria for the outcomes of design research. In *IT Artefact Design & Workpractice Intervention, a Pre-ECIS and AIS SIG Prag Workshop*.
- Cass, A. G., Lerner, A. S., McCall, E. K., Osterweil, L., Jr., S. M. S., and Wise, A. (2000). Little-jil/juliette: a process definition language and interpreter. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 754–757.
- Castro, J., Alencar, F., and Cysneiros, G. (2000). Closing the gap between organizational requirements and object oriented modeling. *Journal of the Brazilian Computer Society*, 7.
- Chen, P. P.-S. (1976). The entity-relationship model toward a unified view of data. *ACM Trans. Database Syst.*, 1:9–36.
- Christiansen, H., Have, C. T., and Tveitane, K. (2007). From use cases to UML class diagrams using logic grammars and constraints. *Proc. of the Recent advances in Natural Language Processing. Bulgaria*.
- Coalition, W. M. (2011). Web site. <http://www.wfmc.org/>. Visited in February 2011.
- Cockburn, A. (2001). *Writing Effective Use Cases*. Addison Wesley.
- Collins-Cope, M. (1999). The requirements/service/interface (RSI) approach to use case analysis (a pattern for structured use case development). In *Technology of Object-Oriented Languages and Systems, 1999. Proceedings of*, pages 172–183.
- Cox, K. (2002). Heuristics for use case descriptions. PhD Thesis, Bournemouth University.
- Cruz, E. F., Machado, R. J., and Santos, M. Y. (2012). From business process modeling to data model: A systematic approach. In *QUATIC 2012, Thematic Track on Quality in ICT Requirements Engineering, IEEE Computer Society Press, Los Alamitos, California, U.S.A.*, pages 205–210.
- Cruz, E. F., Machado, R. J., and Santos, M. Y. (2014a). Derivation of data-driven software models from business process representations. In *9th International Conference on the Quality of Information and Communications Technology (QUATIC2014)*, pages 276–281. IEEE Compute Society.
- Cruz, E. F., Machado, R. J., and Santos, M. Y. (2014b). From business process models to use case models: A systematic approach. In Aveiro, D., Tribollet, J., and Gouveia, D., editors, *Advances in Enterprise Engineering VIII*,

- volume 174 of *Lecture Notes in Business Information Processing*, pages 167–181. Springer International Publishing.
- Cruz, E. F., Machado, R. J., and Santos, M. Y. (2014c). On the decomposition of use cases for the refinement of software requirements. In *Computational Science and Its Applications (ICCSA), 2014 14th International Conference on*, pages 237–240. IEEE - Compute Society.
- Cruz, E. F., Machado, R. J., and Santos, M. Y. (2015a). Bridging the gap between a set of interrelated business process models and software models. In *17th International Conference on Enterprise Information Systems*, pages 338–345.
- Cruz, E. F., Machado, R. J., and Santos, M. Y. (2016). Deriving software design models from a set of business processes. In *4th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2016*.
- Cruz, E. F., Santos, M. Y., and Machado, R. J. (2015b). Deriving a data model from a set of interrelated business process models. In *17th International Conference on Enterprise Information Systems*, pages 49–59.
- Darimont, R. and van Lamsweerde, A. (1996). Formal refinement patterns for goal-driven requirements elaboration. In *SIGSOFT'96 CA, USA*.
- Davis, W. S. and Yen, D. C. (1999). *The systems development life cycle in The Information System Consultant's Handbook: Entity-relationship diagrams*. CRC Press, New York.
- de la Vara, J., Fortuna, M., Sánchez, J., Werner, C., and Borges, M. (2009). A requirements engineering approach for data modelling of process-aware information systems. In Abramowicz, W., editor, *Business Information Systems*, volume 21 of *Lecture Notes in Business Information Processing*, pages 133–144. Springer Berlin Heidelberg.
- de la Vara, J. and Sánchez, J. (2009). BPMN-based specification of task descriptions: Approach and lessons learnt. In Glinz, M. and Heymans, P., editors, *Requirements Engineering: Foundation for Software Quality*, volume 5512 of *LNCS*, pages 124–138. Springer Berlin Heidelberg.
- Debnath, N., Riesco, D., Cota, M. P., and Romero, D. (2006). Supporting the SPEM with a UML extended workflow metamodel. *IEEE*, 1:1151–1154.
- Delgado, A., Calegari, D., Milanese, P., Falcon, R., and García, E. (2015). A systematic approach for evaluating bpm systems: case studies on open source and proprietary tools. In *Open Source Systems: Adoption and Impact*, pages 81–90. Springer Berlin Heidelberg.

- Dietz, J. (2003). Deriving use cases from business process models. In Song, I.-Y., Liddle, S., Ling, T.-W., and Scheuermann, P., editors, *Conceptual Modeling - ER 2003*, volume 2813 of *LNCS*, pages 131–143. Springer Berlin Heidelberg.
- Dietz, J. L. (2001). Demo: Towards a discipline of organisation engineering. *European Journal of Operational Research*, 128(2):351 – 363. Complex Societal Problems.
- Dijkman, R. M., Dumas, M., and Ouyang, C. (2008). Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281 – 1294.
- Dijkman, R. M. and Joosten, S. M. (2002a). An algorithm to derive use cases from business processes. In *6th International Conference on Software Engineering and Applications (SEA)*, pages 679–684.
- Dijkman, R. M. and Joosten, S. M. (2002b). Deriving use case diagrams from business process models. Technical report, CTIT Technical Report, The Netherlands.
- Ditze, A. and Henninger, T. (2010). Methodical approach for business analyst with BPMN. *Modeling Magazine* 5.
- Dobing, B. and Parsons, J. (2000). *Understanding the Role of Use Cases in UML: A Review and Research Agenda.*, volume 11, chapter 8, pages 28–36. *Journal of Database Management (JDM)*.
- Dobing, B. and Parsons, J. (2006). How UML is used. *Communications of the ACM.*, 49:109–113.
- Dobing, B. and Parsons, J. (2009). Dimensions of uml diagram use: practitioner survey and research agenda. *Principle Advancements in Database Management Technologies: New Applications and Frameworks*, pages 271–290.
- Dong, M. and Chen, F. F. (2005). Petri net-based workflow modelling and analysis of the integrated manufacturing business processes. *The International Journal of Advanced Manufacturing Technology*, 26:1163–1172. 10.1007/s00170-004-2089-4.
- Dorn, J., Grün, C., Werthner, H., and Zapletal, M. (2009). From business to software: a B2B survey. *Information Systems and E-Business Management*, 7:123–142. 10.1007/s10257-008-0082-4.
- Drazan, J. and Mencl, V. (2007). Improved processing of textual use cases: Deriving behavior specifications. In van Leeuwen, J., Italiano, G., van der Hoek, W., Meinel, C., and Sack, H., editors, *SOFSEM 2007: Theory and*

Practice of Computer Science, volume 4362 of *Lecture Notes in Computer Science*, pages 856–868. Springer Berlin Heidelberg.

- Eike, B., Wojciech, F., P., H. R., Hanna, K., Pelz, and Elisabeth (1998). M-nets: An algebra of high-level petri nets, with an application to the semantics of concurrent programming languages. *Acta Informatica*, 35:813–857.
- Eriksson, H.-E. and Penker, M. (2000). Business modeling with UML. Technical report, Open Training.
- Estublier, J. (2006). Software are processes too. In Li, M., Boehm, B., and Osterweil, L., editors, *Unifying the Software Process Spectrum*, volume 3840 of *Lecture Notes in Computer Science*, pages 25–34. Springer Berlin Heidelberg.
- Evans, E. (2011). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional.
- Fantechi, A., Gnesi, S., Lami, G., and Maccari, A. (2003). Applications of linguistic techniques for use case analysis. *Requirements Engineering*, 8(3):161–170.
- Fernandes, J. M., Machado, R., Monteiro, P., and Rodrigues, H. (2006). A demonstration case on the transformation of software architectures for service specification. In *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, volume 225 of *IFIP International Federation for Information Processing*, pages 235–244. Springer Boston.
- Fernandes, J. M., Machado, R. J., and Santos, H. D. (2000). Modeling industrial embedded systems with UML. In *Proceedings of the eighth international workshop on Hardware/software codesign*, CODES '00, pages 18–22, New York, NY, USA. ACM.
- Ferreira, N., Santos, N., Machado, R. J., and Gasevic, D. (2012). Derivation of process-oriented logical architectures: An elicitation approach for cloud design. In *PROFES'2012, LNCS Series, Springer-Verlag, Berlin Heidelberg, Germany*.
- Fettke, P. (2009). How conceptual modeling is used. *Communications of the Association for Information Systems*, 25:571–592.
- Fettke, P., Loos, P., and Zwicker, J. (2005). Business process reference models: Survey and classification. In *First International Workshop on Business Process Reference Models (BPRM05)*.
- Fowler, M. (2004). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional.

- Giaglis, G. M. (2001). A taxonomy of business process modeling and information systems modeling techniques. *International Journal of Flexible Manufacturing Systems*, 13:209–228.
- Glinz, M. (2000). Problems and deficiencies of UML as a requirements specification language. In *Proceedings of the 10th International Workshop on Software Specification and Design, IWSSD '00*, Washington, DC, USA. IEEE Computer Society.
- Gomma, H. (2011). *Software modeling and design: UML, Use cases, Patterns, and Software Architectures*. Cambridge University Press.
- Greenfield, J. and Short, K. (2004). Software factories: Assembling applications with patterns, frameworks, models and tools.
- Group, L. P. W. (2006). Little-jil 1.5 - language report. Technical report, LASER Process Working Group.
- Hammer, M. and Champy, J. (2001). Reengineering the corporation: a manifesto for business revolution. *Harper Business*.
- Hausmann, J., Heckel, R., and Taentzer, G. (2002). Detection of conflicting functional requirements in a use case-driven approach. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pages 105 –115.
- Hevner, A. and March, S. (2003). The information systems research cycle. *Computer*, 36(11):111 – 113.
- Hevner, A. R., March, S. T., Jinsoo, P., and Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1):75 – 105.
- Hofreiter, B., Huemer, C., Liegl, P., Schuster, R., and Zapletal, M. (2010). UMM Add-In: A UML Extension for UN/CEFACT Modeling Methodology. Technical report, Vienna University of Technology.
- Hull, E., Jackson, K., and Dick, J. (2011). *Requirements Engineering*. Springer Science & Business Media.
- IEC, I. (2010). Systems and software engineering, systems and software quality, requirements and evaluation (square) system and software quality models. Technical report, The International Organization for Standardization and the International Electrotechnical Commission.
- III, J. G. D. (2014). Aligning customer needs: Business process management (bpm) and successful change management in the l. tom perry special collections. *Library Leadership & Management*, 29(1).

- Ilieva, M. G. and Ormandjieva, O. (2006). Models derived from automatically analyzed textual user requirements. In *Software Engineering Research, Management and Applications*.
- Issa, A. A. (2007). Utilising refactoring to restructure use-case models. In *Proceedings of the World Congress on Engineering, WCE 2007*.
- Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Addison-Wesley.
- Jalote, P. (2008). *A concise Introduction to Software Engineering*. Springer Science & Business Media.
- Juric, M. B. (2015). A hands-on introduction to bpm. ORACLE.
- Kalenkova, A. A., de Leoni, M., and van der Aalst, W. M. (2014). Discovering, analyzing and enhancing BPMN models using prom*. In *Business Process Management-12th International Conference, BPM*, pages 7–11.
- Karakostas, B., Zorgios, Y., and Alevizos, C. (2006). Automatic derivation of BPEL4WS from IDEF0 process models. *Software and Systems Modeling*, 5:208–218.
- Kees van Hee, Olivia Oanea, R. P., Somers, L., and an der Werf, J. M. (2006). Yasper: a tool for workflow modeling and analysis. *Application of Concurrency to System Design, International Conference on*, 0:279–282.
- Kindler, E. (2004). On the semantics of EPCs: A framework for resolving the vicious circle. In Desel, J., Pernici, B., and Weske, M., editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97. Springer Berlin Heidelberg.
- Ko, R. K. L. (2009). A computer scientist’s introductory guide to business process management (bpm). *Crossroads*, 15:4:11–4:18.
- Kocbek, M., Jošt, G., Heričko, M., and Polančič, G. (2015). Business process model and notation: The current state of affairs. *Computer Science and Information Systems*, 1(00):1–35.
- Korherr, B. and List, B. (2006). Aligning business processes and software connecting the UML 2 profile for event driven process chains with use cases and components. *18th Int. Conf. on Advanced Information Systems Engineering CAiSE ’06. Luxembourg*, pages 19–22.
- Lam, V. (2009). Equivalences of BPMN processes. *Service Oriented Computing and Applications*, 3:189–204. 10.1007/s11761-009-0048-5.

- Lange, C., Chaudron, M., and Muskens, J. (2006). In practice: UML software architecture and design description. *Software, IEEE*, 23(2):40 – 46.
- Lerner, B. S., Christov, S., Osterweil, L. J., Bendraou, R., Kannengiesser, U., and Wise, A. (2010). Exception handling patterns for process modeling. *IEEE Transactions on Software Engineering*, 99(RapidPosts):162–183.
- Liang, Y., Tian, J., Hu, S., Song, Y., and Zhang, Y. (2008). A template-based approach for automatic mapping between business process and BPEL process. *Wuhan University Journal of Natural Sciences*, 13:445–449. 10.1007/s11859-008-0413-9.
- Ling, S. and Schmidt, H. (2000). Time petri nets for workflow modelling and analysis. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 3039–3044.
- Linzhang, W., Jiesong, Y., Xiaofeng, Y., Jun, H., Xuandong, L., and Guoliang, Z. (2004). Generating test cases from UML activity diagram based on gray-box method. In *Software Engineering Conference, 2004. 11th Asia-Pacific*, pages 284–291.
- List, B. and Korherr, B. (2006). An evaluation of conceptual business process modelling languages. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC06, pages 1532–1539, New York, NY, USA. ACM.
- Lubke, D., Schneider, K., and Weidlich, M. (2008). Visualizing use case sets as BPMN processes. In *Requirements Engineering Visualization*, pages 21–25. IEEE.
- Machado, R., Fernandes, J., Monteiro, P., and Rodrigues, H. (2005). Transformation of UML models for service-oriented software architectures. In *Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the*, pages 173 – 182.
- Machado, R., Fernandes, J. a., Monteiro, P., and Rodrigues, H. (2006). Refinement of software architectures by recursive model transformations. In Münch, J. and Vierimaa, M., editors, *Product-Focused Software Process Improvement*, volume 4034 of *LNCS*, pages 422–428. Springer Berlin Heidelberg.
- Magnani, M. and Montesi, D. (2009). BPDMMN: A conservative extension of BPMN with enhanced data representation capabilities. *arXiv preprint arXiv:0907.1978*.
- March, S. T. and Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, 15(4):251 – 266.

- March, S. T. and Storey, V. C. (2008). Design science in the information systems discipline: An introduction to the special issue on design science research. In *Design Science in the Information Systems*, volume 32, pages 725–730. MIS Quarterly.
- Marshall, C. (1999). *Enterprise Modeling with UML: Designing Successful Software through Business Analysis*. Addison-Wesley.
- Martinez, A., Castro, J., Pastor, O., and Estrada, H. (2003). Closing the gap between organizational modeling and information system modeling. *Paper presented at the WER03-VI*.
- Metz, P., O'Brien, J., and Weber, W. (2001). Against use case interleaving. In Gogolla, M. and Kobryn, C., editors, *UML The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, volume 2185 of *LNCS*, pages 472–486. Springer Berlin Heidelberg.
- Meyer, A., Pufahl, L., Fahland, D., and Weske, M. (2013). Modeling and enacting complex data dependencies in business processes. In Daniel, F., Wang, J., and Weber, B., editors, *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 171–186. Springer Berlin Heidelberg.
- Meyer, A., Smirnov, S., and Weske, M. (2011). *Data in business processes*. Universitätsverlag Potsdam.
- Mili, H., Jaoude, G. B., Éric Lefebvre, Tremblay, G., and Petrenko, A. (2003). Business process modeling languages: Sorting through the alphabet soup. In *OOF 22 NO. IST-FP6-508794 (PROTCURE II) September*.
- Monsalve, C., April, A., and Abran, A. (2012). On the expressiveness of business process modeling notations for software requirements elicitation. In *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pages 3132–3137.
- Muehlen, M. and Recker, J. (2008). How much language is enough? theoretical and practical use of the business process modeling notation. In Bellahsene, Z. and Leonard, M., editors, *Advanced Information Systems Engineering*, volume 5074 of *Lecture Notes in Computer Science*, pages 465–479. Springer Berlin Heidelberg.
- Nawrocki, J., Nedza, T., Ochodek, M., and Olek, L. (2006). Describing business processes with use cases. In *BIS*, pages 13–27.
- Olivé, A. (2010). *Conceptual Modeling of Information Systems*. Springer Science & Business Media.

- OMG (2008). Software & systems process engineering meta-model specification, version 2.0. Technical report, Object Management Group.
- OMG (2010a). BPMN 2.0 by example. Technical report, Object Management Group.
- OMG (2010b). Object constraint language. Technical report, Object Management Group.
- OMG (2011a). Business process model and notation (BPMN), version 2.0. Technical report, Object Management Group.
- OMG (2011b). Meta object facility (MOF) 2.0 query/view/transformation specification. Technical report, Object Management Group.
- OMG (2011c). OMG meta object facility (MOF) core specification. Technical report, Object Management Group.
- OMG (2012). Unified modeling language (OMG UML), version 2.5. Technical report, Object Management Group.
- ORACLE (2011). <http://blogs.oracle.com/bpmbrasil/bpm/>. Web site. Accessed January 27, 2011.
- Osterwalder, A. and Pigneur, Y. (2010). *Business model generation: a handbook for visionaries, game changers, and challengers*. John Wiley & Sons.
- Oswald, H., Esser, R., and Mattmann, R. (1990). An environment for specifying and executing hierarchical petri nets. In *Proceedings of the 12th international conference on Software engineering*, pages 164–172. IEEE Computer Society Press.
- Ouyang, C., Dumas, M., Breutel, S., and ter Hofstede, A. (2006). Translating standard process models to BPEL. In Dubois, E. and Pohl, K., editors, *Advanced Information Systems Engineering*, volume 4001 of *Lecture Notes in Computer Science*, pages 417–432. Springer Berlin Heidelberg.
- Paradkar, A. M. and Sinha, A. (2015). Deriving process models from natural language use case models. US Patent 8,949,773.
- Peppers, K., Tuunanen, T., Gengler, C. E., Rossi, M., Hui, W., Virtanen, V., and Bragge, J. (2006). The design science research process: a model for producing and presenting information systems research. In *Proceedings of the first international conference on design science research in information systems and technology (DESRIST 2006)*, pages 83–106.

- Phalp, K., Vincent, J., and Cox, K. (2007). Improving the quality of use case descriptions: empirical assessment of writing guidelines. *Software Quality Journal*, 15(4):383–399.
- Pilone, D. and Pitman, N. (2005). *UML 2.0 in a nutshell*. O’Reilly.
- Polancic, G. (2015). What makes bpmn complex? *BPMN Series ATL001:15*.
- Pons, C. and Kutsche, R.-D. (2004). Traceability across refinement steps in UML modeling. In *UML Workshop in Software Model Engineering WiSME*.
- Quartel, D., Pires, L., Franken, H., and Vissers, C. (1995). An engineering approach towards action refinement. In *Distributed Computing Systems, 1995., Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of*, pages 266–273.
- Raedts, I., Petkovic, M., Usenko, Y. S., van der Werf, J. M., Groote, J. F., and Somers, L. (2007). Transformation of BPMN models for behaviour analysis. *MSVVEIS*.
- Recker, J. C. (2008). BPMN Modeling – Who, Where, How and Why. *BPTrends*, 5(3):1–8.
- Redlich, D., Blair, G., Rashid, A., Molka, T., and Gilani, W. (2014). Research challenges for business process models at run-time. In Bencomo, N., France, R., Cheng, B., and Abmann, U., editors, *Models at run.time*, volume 8378 of *Lecture Notes in Computer Science*, pages 208–236. Springer International Publishing.
- Regnell, B., Andersson, M., and Bergstrand, J. (1996). A hierarchical use case model with graphical representation. In *Engineering of Computer-Based Systems, 1996. Proceedings., IEEE Symposium and Workshop on*, pages 270–277.
- Rittgen, P. (2008). From business process model to information systems model: Integrating DEMO and UML. In *Modern Systems Analysis and Design Technologies and Applications*.
- Rodríguez, A., de Guzmán, I. G.-R., Fernández-Medina, E., and Piattini, M. (2010). Semi-formal transformation of secure business processes into analysis class and use case models: An MDA approach. *Information and Software Technology*, 52(9):945 – 971.
- Rodríguez, A., Fernández-Medina, E., and Piattini, M. (2007). Towards CIM to PIM transformation: From secure business processes defined in BPMN to use-cases. In *Business Process Management*, pages 408–415.

- Rodríguez, A., Fernández-Medina, E., and Piattini, M. (2008). Towards obtaining analysis-level class and use case diagrams from business process models. In *Advances in Conceptual Modeling Challenges and Opportunities*, volume 5232 of *Lecture Notes in Computer Science*, pages 103–112. Springer Berlin Heidelberg.
- Rolland, C. and Achour, C. B. (1998). Guiding the construction of textual use case specifications. *Data & Knowledge Engineering*, 25:125 – 160.
- Roussev, B. (2003). Generating ocl specifications and class diagrams from use cases: A newtonian approach. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pages 10–pp. IEEE.
- Rungworawut, W. and Senivongse, T. (2005). A guideline to mapping business process to UML class diagrams. In *WSEAS Transactions on Computer*, volume 4, pages 526–1533.
- Rungworawut, W. and Senivongse, T. (2006). Using ontology search in the design of class diagram from business process model. In *World Academy of science, Engineering and Technology*, volume 12, pages 165–170.
- Russell, N., van der Aalst, W. M. P., ter Hofstede, A. H. M., and Wohed, P. (2006). On the suitability of UML 2.0 activity diagrams for business process modelling. In *Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling - Volume 53*, APCCM '06, pages 95–104, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.
- Samarasinghe, N. and Somé, S. S. (2005). Generating a domain model from a use case model. In *Intelligent and adaptive systems and software engineering*.
- Santos, M. Y. and Machado, R. J. (2010). On the derivation of class diagrams from use cases and logical software architectures. In *2010 Fifth International Conference on Software Engineering Advances*.
- Savié, D. and da Silva, A. R. (2012). Use case specification at different levels of abstraction. In *2012 Eighth International Conference on the Quality of Information and Communications Technology*.
- Scheer, A.-W. and Nüttgens, M. (2000). *ARIS architecture and reference models for business process management*. Springer Science & Business Media.
- Schmiedel, T. and vom Brocke, J. (2015). Business process management: Potentials and challenges of driving innovation. In vom Brocke, J. and Schmiedel, T., editors, *BPM - Driving Innovation in a Digital World*, Management for Professionals, pages 3–15. Springer International Publishing.

- Scott, J. E. (2007). Mobility, business process management, software sourcing, and maturity model trends: Propositions for the is organization of the future. *Information Systems Management*, 24(2):139–145.
- Sendall, S. and Kozaczynski, W. (2003). Model transformation the heart and soul of model-driven software development. Technical report, Swiss Federal Institute of Technology in Lausanne (EPFL).
- Shishkov, B., Xie, Z., Liu, K., and Dietz, J. L. (2002). Using norm analysis to derive use cases from business processes. In *Proceedings of the 5th Workshop On Organizational Semiotics*.
- Silingas, D. and Butleris, R. (2008). UML-intensive framework for modeling software requirements. In *Proceedings of International Conference on Information Technologies*, pages 334–342.
- Silva, A. and Videira, C. (2005). *UML Metodologias e ferramentas CASE*, volume 1. CentroAtlantico.pt, 2^a edition.
- Štolfa, S. and Vondrák, I. (2008). Mapping from business processes to requirements specification. *Retrieved on 7th Aug*.
- Sturm, A. (2008). Enabling off-line business process analysis: A transformation-based approach. *Proceedings of BPMDS*, 8:67.
- ter Hofstede, A., van der Aalst, W., and Weske, M. (2003). Business process management: A survey. In Weske, M., editor, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg.
- Tiwari, S. and Gupta, A. (2015). A systematic literature review of use case specifications research. *Information and Software Technology*, 67:128 – 158.
- Truscan, D., Fernandes, J. M., and Lilius, J. (2004). Tool support for DFD-UML model-based transformations. In *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*, pages 388 – 397.
- Vaishnavi, V. K. and Jr., W. K. (2008). *Design Science Research Methods and Patterns Innovating Information and Communication Technology*. Auerbach Publications.
- van der Aa, H., Leopold, H., Mannhardt, F., and Reijers, H. (2015). On the fragmentation of process information: Challenges, solutions, and outlook. In Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., and Ma, Q., editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 214

- of *Lecture Notes in Business Information Processing*, pages 3–18. Springer International Publishing.
- van der Aalst, W. (1999). Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650.
- van der Aalst, W. (2004). Business process management demystified: A tutorial on models, systems and standards for workflow management. In Desel, J., Reisig, W., and Rozenberg, G., editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 1–65. Springer Berlin Heidelberg.
- van der Aalst, W. (2015). Business process simulation survival guide. In vom Brocke, J. and Rosemann, M., editors, *Handbook on Business Process Management 1*, International Handbooks on Information Systems, pages 337–370. Springer Berlin Heidelberg.
- van der Aalst, W. M. (2003). Patterns and XPDL: A critical evaluation of the XML process definition language. Technical report, Department of Technology Management Eindhoven University of Technology, The Netherlands.
- Venable, J. R. (2006). The role of theory and theorising in design science research. *First International Conference on Design Science Research in Information Systems and Technology, Claremont, California.*, 1:1–18.
- Weber, B., Reichert, M., Mendling, J., and Reijers, H. A. (2011). Refactoring large process model repositories. *Computers in Industry*, 62(5):467–486.
- Weske, M. (2012). *Business Process Management Concepts, Languages, Architectures*. Springer Science & Business Media.
- Weske, M., van der Aalst, W., and Verbeek, H. (2004). Advances in business process management. In *Data & Knowledge Engineering*, 50.
- White, S. A. and Miers, D. (2008). *BPMN Modeling and Reference Guide*. Future Strategies Inc.
- Whittle, J. and Jayaraman, P. (2006). Generating hierarchical state machines from use case charts. In *Requirements Engineering, 14th IEEE International Conference*, pages 19–28.
- Wise, A., Cass, A. G., Lerner, B. S., and McCall, E. K. (2000). Using little-jil to coordinate agents in software engineering. Technical report, HP Laboratories and IBM T. J. Watson Research Center.

- Wise, A., Lerner, B. S., McCall, E. K., Osterweil, L. J., and Jr, S. M. S. (1999). Specifying coordination in processes using little-jil. *Department of Computer Science, University of Massachusetts at Amherst, Technical Report*, pages 99–71.
- Wohed, P., van der Aalst, W., Dumas, M., ter Hofstede, A., and Russell, N. (2006). On the suitability of BPMN for business process modelling. In Dustdar, S., Fiadeiro, J., and Sheth, A., editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 161–176. Springer Berlin Heidelberg.
- Ying and Liang (2003). From use cases to classes: a way of building object model with UML. *Information and Software Technology*, 45(2):83 – 93.
- Yue, T., Briand, L., and Labiche, Y. (2009). A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation. In Schurr, A. and Selic, B., editors, *Model Driven Engineering Languages and Systems*, volume 5795 of *LNCS*, pages 484–498. Springer Berlin Heidelberg.
- Yue, T., Briand, L., and Labiche, Y. (2011). A systematic review of transformation approaches between user requirements and analysis models. *Requirements Engineering*, 16:75–99.
- Zowghi, D. and Coulin, C. (2005). Requirements elicitation: A survey of techniques, approaches, and tools. In Aurumand, A. and Wohlin, C., editors, *Engineering and Managing Software Requirements*, pages 19–46. Springer Berlin Heidelberg.

Appendix A

Use Cases Model of the Library Demonstration Case

Use case Name	Use case Description
{U1.1} Check Borrower Documents	Actors: Borrower, Attendant Trigger: Borrower wants to register Scenario: Receives documents from <Borrower>.
{U1.2} Stores Borrower	Actors: Borrower, Attendant Pre-condition: Are Borrower documents OK? Is Yes. Scenario: Writes information on <Borrower>. Sends register confirmation to <Borrower>.
{U1.3} Reject Registration	Actors: Borrower, Attendant Pre-condition: Are documents OK? Is No. Scenario: Sends message <documents not OK> to <Borrower>.
{U2.1} Check Borrower Identification	Actors: Borrower, Attendant Trigger: The message request a book arrives from <Borrower>. Scenario: Receives Borrower identification from <Borrower>. Reads information from <Borrower>.
{U2.2} Check Book status	Actors: Borrower, Attendant Pre-condition: Borrower is register as member? is yes. Scenario: Receives Book information from <Borrower>. Reads information from <Book>.

Use case Name	Use case Description
{U2.3} Register Loan	Actors: Borrower, Attendant Pre-condition: Is book available? is yes. Scenario: Writes information on <Loan>. Sends loan information to <Borrower>.
{U.2.4} Deny Loan	Actors: Borrower, Attendant Pre-condition: Borrower is register as member? is No. Scenario: Sends message you must be registered to <Borrower>.
{U2.5} Inform book not available	Actors: Borrower, Attendant Pre-condition: Is book available? is no. Scenario: Sends message book not available to <Borrower>.
U3.1 Receives reservation request	Actors: Borrower, Attendant Trigger: The event start process occurs. Scenario: Receives the message I want to reserve a book from <Borrower>.
{U3.2} Check Book	Actors: Attendant Pre-condition: Is borrower registered as a member? Is yes. Scenario: Reads information from <Book>.
{U3.3} Check Borrower	Actors: Attendant Pre-condition: Receives reservation request is completed. Scenario: Reads information from <Borrower>.
{U3.4} Add reservation	Actors: Borrower, Attendant Pre-condition: Is it possible to reserve the Book? is Yes. Pos-condition: The event End process is created. Scenario: Writes information on <Reservation>. Send the message Book is reserved to <Borrower>.
{U3.5} Deny reservation	Actors: Borrower, Attendant Pre-condition: Is it possible to reserve the Book? is No. Scenario: Send the message You must be registered to <Borrower>. Send message book is not available to <Borrower>.
{U4.1} Receive returned book	Actors: Attendant Trigger: The message return a book arrives from <Borrower>.

Use case Name	Use case Description
{U4.2} Check for delivery delay	Actors: Attendant Pre-condition: Search for book's active loan is completed. Scenario: Reads information from <Loan>.
{U4.3.1} calculate fine	Actors: Borrower, Attendant Pre-condition: Get loan info is completed. Scenario: Send value of the fine to <Borrower>.
{U4.3.2} Collects payment	Actors: Borrower, Attendant Pre-condition: Calculate fine is completed. Scenario: Receives payment from <Borrower>.
{U4.3.3} Pass receipt	Actors: Borrower, Attendant Pre-condition: Collects payment is completed. Scenario: Writes information on <Receipt>. Send receipt to <Borrower>.
{U4.3.4} Get loan info	Actors: Attendant Pre-condition: The delivery date has been exceeded? is yes. Scenario: Reads information from <Loan>.
{U4.4} Update Loan info	Actors: Borrower, Attendant Pre-condition: The Penalty treatment is completed OR The delivery date has been exceeded? is no. Pos-condition: The event End process is created. Scenario: Writes information on <Loan>.
{U4.5} Search for book's active loan	Actors: Attendant Pre-condition: The receive returned book is completed. Scenario: Reads information from <Loan>.
{U5.1} Receive renew loan request	Actors: Borrower, Attendant Trigger: The message I want to renew a loan arrives from <Borrower>. Scenario: Receives Loan information from <Borrower>.

Use case Name	Use case Description
{U5.2} Check for reservations	Actors: Attendant Pre-condition: The receive loan info is completed. Scenario: Reads information from <Reservation>.
{U5.3} Check delivery date	Actors: Attendant Pre-condition: Is the Book reserved? is No. Scenario: Reads information from <Loan>.
{U5.4} Loan not renewed	Actors: Borrower, Attendant Pre-condition: Is the Book reserved? is Yes OR Is delivery date exceeded? is Yes. Pos-condition: The event End process is created. Scenario: Send message can't renew Loan to <Borrower>.
{U5.5} Update delivery date	Actors: Borrower, Attendant Pre-condition: Is the delivery date exceeded? is NO Pos-condition: The event End process is created. Scenario: Writes information on <Loan>. Sends new delivery date to <Borrower>.

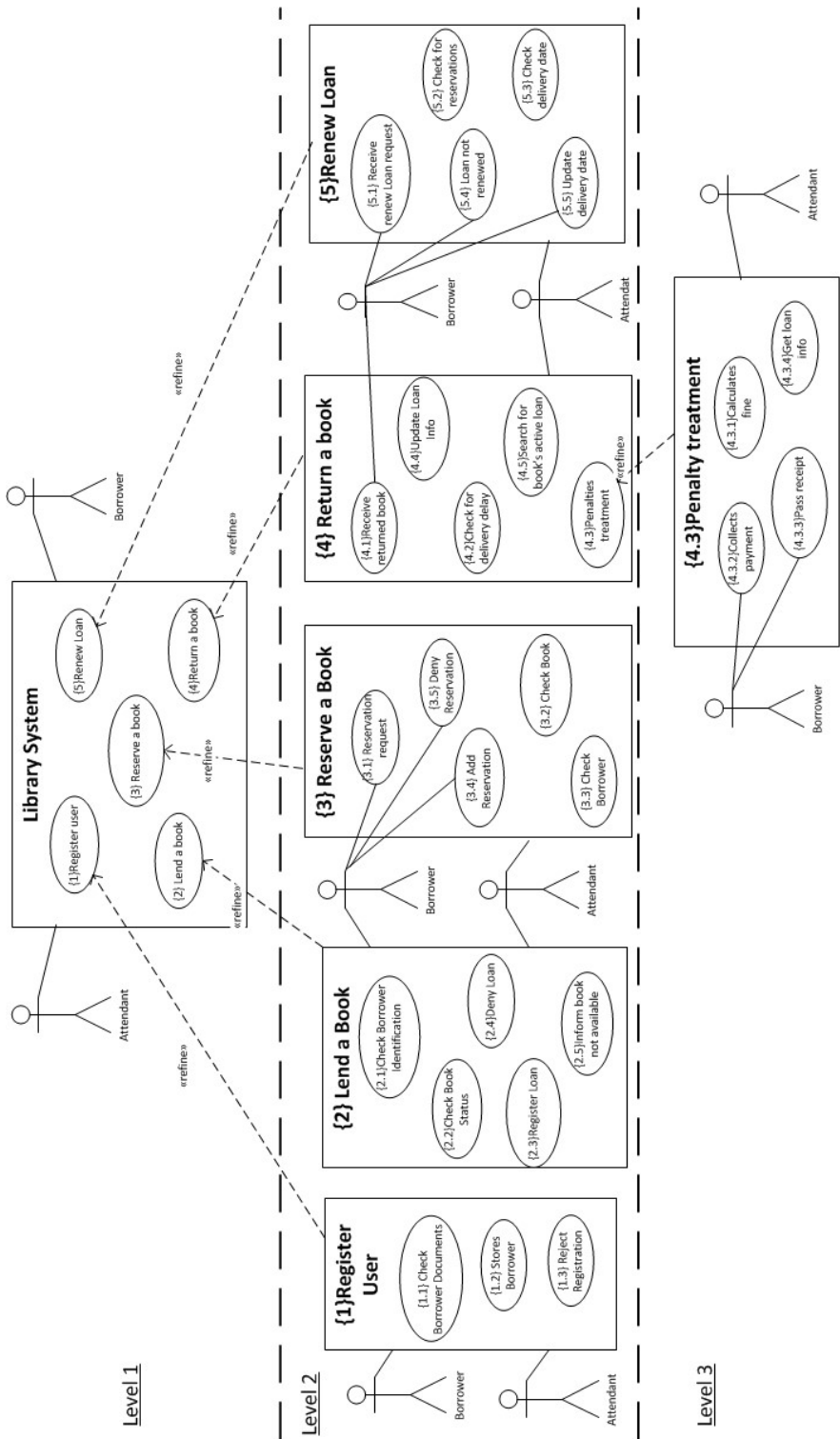


Figure A.1: A use case diagram of the Library Demonstration Case