DEMO Models Based Automatic Workflow Process Generation

MASTER DISSERTATION

Carlos Alberto da Silva Figueira
MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

september | 2013

DEMO Models Based Automatic Workflow Process Generation

MASTER DISSERTATION

Carlos Alberto da Silva Figueira
MASTER IN INFORMATICS ENGINEERING

SUPERVISOR
David Sardinha Andrade de Aveiro

DEMOBAKER

***DE**mo **MO**dels **B**ased **A**utomatic wor**K**flow proc**E**ss gene**R**ation*

Geração de processos automáticos de workflow baseados em modelos DEMO

Master's thesis in Informatics Engineering

by

Carlos Figueira

30th September 2013

University of Madeira - Competence Center of Exact Sciences and Engineerings

www.uma.pt

Madeira Interactive Technologies Institute

www.m-iti.org

Author

Candidate	Carlos Alberto da Silva Figueira
Student Number	2055005
Email	carlosafigueira@gmail.com

Thesis Title

DEMO models based automatic workflow process generation

Advisor

Professor Doutor David Sardinha Andrade de Aveiro

Abstract

Nowadays, more than half of the computer development projects fail to meet the final users' expectations. One of the main causes is insufficient knowledge about the organization of the enterprise to be supported by the respective information system. The DEMO methodology (Design and Engineering Methodology for Organizations) has been proved as a well-defined method to specify, through models and diagrams, the essence of any organization at a high level of abstraction. However, this methodology is platform implementation independent, lacking the possibility of saving and propagating possible changes from the organization models to the implemented software, in a runtime environment. The Universal Enterprise Adaptive Object Model (UEAOM) is a conceptual schema being used as a basis for a wiki system, to allow the modeling of any organization, independent of its implementation, as well as the previously mentioned change propagation in a runtime environment. Based on DEMO and UEAOM, this project aims to develop efficient and standardized methods, to enable an automatic conversion of DEMO Ontological Models, based on UEAOM specification into BPMN (Business Process Model and Notation) models of processes, using clear semantics, without ambiguities, in order to facilitate the creation of processes, almost ready for being executed on workflow systems that support BPMN.

Keywords

DEMO, BPMN, BPM, information systems, action model, meta model

Resumo

Atualmente, mais de metade dos projetos de desenvolvimento de software falham em alcançar as expetativas dos seus utilizadores finais. Uma das causas principais está na falta de conhecimento da organização da empresa para a qual se pretende a construção do sistema. A metodologia DEMO (Design and Engineering Methodology for Organizations) está provada como sendo uma forma eficaz de especificar através da construção de diagramas e modelos, a essência de qualquer organização, a um nível alto de abstração. No entanto, esta metodologia é independente de qualquer plataforma de implementação, falhando na possibilidade de atualização e propagação de possíveis alterações dos modelos da organização para o software implementado, em modo de execução. O Universal Enterprise Adaptive Object Model (UEAOM) é um esquema conceptual usado como base de um sistema baseado em páginas wiki que permitirá modelar uma qualquer organização, independentemente da sua implementação, bem como a já mencionada, propagação de alterações em modo de execução. Com base no DEMO e no UEAOM, pretende-se com a realização deste projeto, desenvolver métodos eficientes e padronizados que possibilitem a conversão automática de Modelos Ontológicos (DEMO) baseados na especificação UEAOM, em modelos de processos na notação BPMN (Business Process Model and Notation) usando-se uma semântica clara e sem ambiguidades, facilitando-se a criação de processos praticamente prontos a executar em sistemas de workflow que suportem BPMN.

Palavras-chave

DEMO, BPMN, BPM, sistemas de informação, modelo de ação, meta modelo

Acknowledgements

Acknowledgments are always a tricky part of any long term work due to the fact of, and normally without realizing, forgetting important parts involved, in some cases important parts. For those, who feel and truly know that were an important part of my path along this important phase of my life, I am eternally and truly grateful and you would be aware that without your existence, support and those wise words on the right moments, I would not be at this phase, writing down this words, thinking exactly about you all. You all, were part of this important journey, so that, and to all of you my dear friends, thank you!

This master thesis and its results were possible thanks to my advisor, Professor Doutor David Sardinha Andrade de Aveiro and his support to guide me in the right directions through the entire project. I also thank to University of Madeira and the Madeira Interactive Technologies Institute (M-ITI), which provided the adequate conditions to accomplish the project.

Regarding my family, I would be eternally grateful to my parents, Alberto Figueira and Maria Figueira, who gave me the possibility, the courage, the strength and the financial support to keep me only concentrated in the development of this project. To my siblings, Flávio e Diogo Figueira who gave me a huge and constant support along the way, to keep me going, fighting and trying, even when the results were not the expected. For my better-half, Sabina Silva, for always being by my side, no matter what.

Contents

List of Figures

List of Tables

Abbreviations

AOM	Adaptive Object Model
AM	Action Model
BOS	Bonita Open Solution
BNF	Backus-Naur Form
BPML	Business Process Management Language
BPM	Business Process Management
BPQL	Business Process Query Language
BPMN	Business Process Model and Notation
CM	Construction Model
DEMO	Design and Engineering Methodology for Organizations
DMM	Demo Meta-model
DPM	Demo Process Model
GPL	Gnu General Public License
JBPM	Java Business process management
OMG	Object Management Group
PM	ProcessMaker
PHP	Hypertext Preprocessor
SM	State Model
SOA	Service-oriented Architecture
TRT	Transaction Result Table
UEAOM	Universal Enterprise Adaptive Object Model
XML	Extensible Markup Language
WYSIWYG	What you see is what you get
WOSL	World Ontology Specification Language

CHAPTER 1

Introduction

Within the computer development project communities it is common sense that research and development projects does not end up properly without a well defined plan, good and valid research methods and the right approach to it. It is also true that even with a good plan it is easy to make a possible implementation difficult to understand and transmit to further readers. This seems counterintuitive, however it is plan of this thesis report, to narrow down the gap between a plan, its related development and its consequent implementation.

This thesis report is hoped to be an appropriated plan and an excellent support, to correctly understand the research that take place along this project.

1.1 Motivation

From [1], where case studies were made, a survey [2] [3] made to about 800 Information Technologies (IT) managers, results that 63% of total software development projects fails, 49% suffered budget overruns, 47% had higher than expected maintenance costs and 41% failed to deliver the expected business value and final users' expectations. From these case studies, was found that the some of the common causes of software failures are based on the lack of clear, well-thought-out goals and specifications, poor management and poor communication among costumers, designers and programmers, [4] unrealistically low budget requests, and underestimates of time requirements, use of new technology maybe for which the software developers do not have adequate experience and expertise and refusal to recognize or admit that a

project is in trouble [1].

Based on these studies and their reality, new ways of capturing the essential information about how an organization really work and what they really need, are extremely important to the software development field.

The Organizational Engineering field of study appeared on the nineteens and comes to add new concepts to the way organizations are understood and new ways to analyze their reality, always regardless their implementation.

The DEMO methodology, a well known defined method used to model organizations is the result of the application of the Organizational Engineering field of study. Its application, results in four models: the Construction Model, Process Model, Action Model and the State Model. The major value of these models, besides being easily to understand, are by being independent of the organization final implementation and that, is extremely valuable in terms of what really matters in the way how enterprises are organized. However, and for this particular thesis, was concluded that some implementation details would be relevant and important to be included. Bearing that in mind, was thought as being valuable for this thesis finding a way to include some of those details on the Action Model, more precisely with the creation of the Action Rules[1].

The Universal Enterprise Adaptive Object Model (UEAOM) is a conceptual schema being used as the base for the effort to create a wiki-based system that allow an effective enterprise modeling, independently of the language used, so that all the organization changes can be made, saved and propagated in runtime environment in a consistent and coherent way. Having all the DEMO models instantiated on the wiki-system all that information can be gathered and worked out for other type of outputs, as a construction of an automatic workflow process based on the DEMO models and diagrams of an enterprise.

The BPMN (Business Process Model and Notation) as a standard notation of the BPM process constructions, is an excellent method to have the DEMO models represented, constructed and finally executed on a workflow system that supports BPMN.

These some news ways of how organizations are viewed and represented by using some well defined methodologies, as DEMO, UEAOM and BPMN opens a new whole of possibilities for even more automated ways of creating processes that work as an excellent support, for even more reliable informations systems, effectively narrowing down the gap that exists between the users' expectations and the final result.

1 Action Rules are part of the Action Model, result of the DEMO methodology application and is reviewed and explained on Sub-Chapter 2.2-DEMO Methodology

1.2 Problem statement

As time passes the world takes a path of evolution and breath taking discovering that sometimes led to inconsistencies between organizational processes and its information systems. Enterprises get bigger, more complex, globalized and integrated, that they must re-think and re-engineer all their way of working. Organizational processes became huge and complex, with more people involved and even bigger cooperation between them in sensible decisions that must be traceable and in constant updating. Flexibility and constant interoperability are nowadays crucial facts to maintain organizations in the right path. In order to do that, since the first use of a technology system, methodologies were used and today even more.

DEMO as a result of the Organizational Engineering field of study has been a growing methodology for the past few years that have been proving along the way to be a well defined and a trustworthy method to model the essence of enterprises, on a high level of abstraction, independent from its real implementation, distinguishing the business, informational and documental actions. From the DEMO application, results four related aspect models: The Construction Model, which represents the organizational construction, the Process Model, that represents the inter-related within and between the organizational transactions, the Action Model, that have all the action rules related to each transaction and its actors and the State Model, that represents the possible states of the production world and the coordination world of an enterprise [5]. Even though and despite the advantages of using DEMO, it lacks on the possibility of a direct implementation of its concise results. DEMO is made to be read and easily understood by humans so it can not be technically and directly implemented by an information system without something in between.

The UEAOM (Universal Enterprise Adaptive Object Model) is an effort to construct a wiki-based system that will gather all the models and diagrams, result of the appliance of the DEMO methodology in wiki-pages, easily editable and updatable.

BPMN is a standard and a well known notation of the Business Process Management methods. Both DEMO and BPM methods are well defined and it certain adds major value to any organization and even more with the combination of both methods.

It is aim of this dissertation project, to create a standardized and well-defined method to enable an automatic conversion of DEMO Ontological Models, based on UEAOM specification into BPMN (Business Process Model and Notation) models of processes, using clear semantics, without ambiguities, in order to facilitate the creation of processes, almost ready to be executed on workflow systems that support BPMN.

1.3 Research approach

The main research idea is based on what is depicted on Figure 1. The first step is about acknowledging BPM frameworks.This step happens in a moment

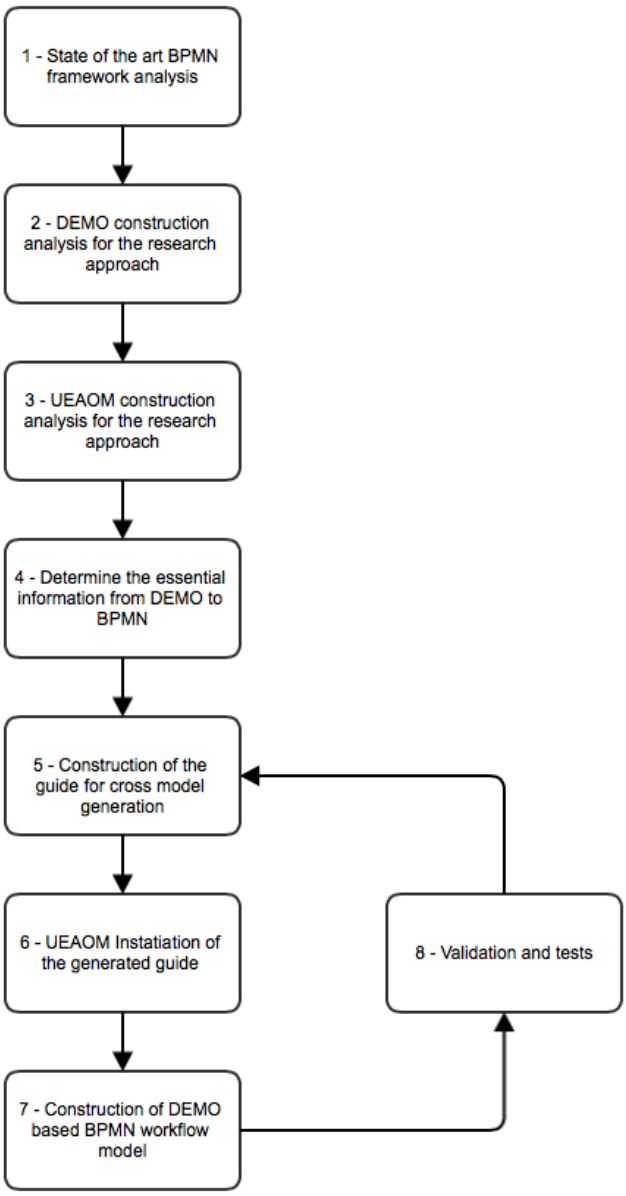


Figure 1: Research approach

where the main goal proposed for this project was the creation of a capable parser to directly convert an UEAOM DEMO model based of an organization into an executable BPMN process, so that a thorough analysis within the BPM

frameworks was due.

After step 1, follows the acknowledgment of DEMO models and how they are constructed. This step is extremely important since it is based on the DEMO models that the final BPMN process is constructed. A full understanding of the DEMO construction works as a solid preparation for the following research.

After accomplishing the previous step, the UEAOM model principles and also its construction details are acknowledged. The DEMO models are represented on this model, so it is a concept that needs to be very clear.

Knowing how DEMO models are constructed it is necessary to understand which are the diagrams that have the essential information to be gathered in order to have the necessary information for the conversion. That part is seen on step 4. Since both, DEMO and BPMN models are conceptually different even in its constructions, how are they going to be compliant? In step 5 were found the conversion rules to make both models compliant.

Steps 5, 6, 7 and 8, work as an iteration cycle. Starts with having the generation guide worked out, then with the instantiation requirements for the guidelines created on the previous step and finally with the BPMN construction. Then the BPMN model is tested in order to check if the models are compliant. The final step is to have a BPMN workflow model compliant with the organization DEMO models.

1.4 Research method

According to A. R. Hevner [6] [7], Design Science Research – the Information Systems Research paradigm that was adopted on the evolving work that took

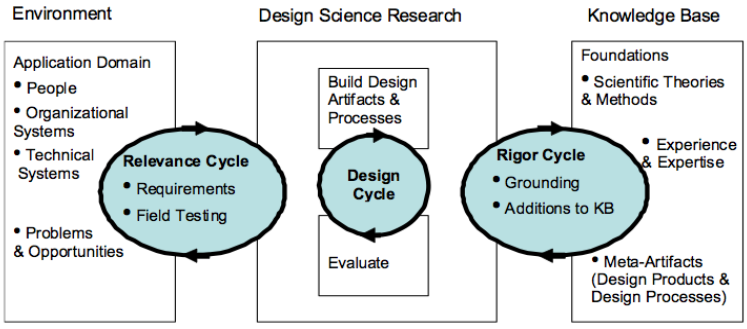


Figure 2: Design science research cycles

Source: From [6]

place in this project – should be seen as a group of three closely related cycles of activities. These activities are depicted on Figure 2. Hevner claims that the individual application of these three activities in an isolated way does not constitute a good design science research. Only the conjunction of the three can actually render good design science research with a valid output. In this thesis, and regarding the relevance cycle depicted on Figure 2, was identified a

clear problem of ambiguity and lack of concise and essential information on current DEMO's action rule syntax – detailed later on Sub-chapter 5.3-New DEMO Action Rule Syntax proposal - so an opportunity to reach a new more solid and concise syntax was at hand. Regarding the Rigor cycle, this project was supported by all the theoretical foundations grounding on DEMO as well as the UEAOM patterns. The most important cycle is the Design cycle itself, out of which resulted the proposal[2] presented on this thesis of a new meta-model for DEMO's Action Model. An exhaustive and thorough evaluation was made with many iterations of the cycle where new elements would be added to DEMO's Action Meta Model and instantiated on the new syntax within the EU-rent[3] case and verifying if it allowed to specify maximum ontological information in a concise and comprehensive way, normally not the case in DEMO's Action Model. While instantiating and increasing the complexity of EU-rent case's action rules to a more realistic level, sometimes was found that some concept in the meta-model should be unary, some other times other concepts should be binary, and at other times was found that would need to specify new concepts at meta-level like *atomic action* and *flow*.

The final proposal presented is the result of a long and thorough process of conceptual evolution and comprehensive instantiation, thus following the tenets of Design Science Research.

1.5 Report structure

Chapter 1, is dedicated to give an insight about this thesis research problem and suggested solution. In Chapter 2, is shown and explained the most important foundations and useful background information about Enterprise Ontology, the PSI-Theory and the Organization Theorem. DEMO methodology is also acknowledged along with the WOSL (World Ontology Specification Language), a specification language proposed by Dietz [8]. The BPM fundamentals and its standard notation - BPMN - is introduced and described. The UEAOM is also reason of attention in this chapter, acknowledging its main foundations, principles and its way of being an important asset for this project.

Is in Chapter 3 where a thorough research and comparison among BPM frameworks is made. One of them is chosen in order to construct and evaluate the final solution model. Implementation examples in each of the frameworks and a final conclusion are described.

Chapter 4 contains the main work of this research. From DEMO to Workflow, describes the necessary requirements from both DEMO and BPMN models, the generation guidelines, how to capture the essential information from

2 As result of the thorough application of this research method a new action rule syntax was proposed and can be seen on Sub-chapter 5.3-New DEMO Action Rule Syntax proposal

3 The EU-Rent case is described on Sub-chapter 5.1-Eu-Rent case description

DEMO models and the necessary conversion rules to make both models compliant.

Throughout Chapter 5, the Workflow Generation evaluation take place. The final BPMN model is constructed based on a real example so its result can be evaluated and checked according the guidelines and respective conversion rules.

CHAPTER 2

Background

In this chapter and without much further deepen on the primordial subject of this thesis, is appropriated to give an insight and enough foundations about the subjects that are used and discussed on this graduation thesis so that the reader can easily focus his mind and effort on the right related subject.

2.1 Enterprise Ontology

Enterprise Ontology is a relatively new theme. Being recent for the most of the ones related to computer technologies, it is important that its definition should be simple, clear and easily understandable.

This novel theme is important for this work since it is based on it that some implementation and important decisions are based on. Before its complete definition there is first a relatively simple question that deserves an answer: what means Ontology? The definition adopted in this work is based on the computer technology related philosophy [9]. Ontology is a combination of two greek words: *onto* and *logia. Onto* stands for "being: that, which is" and *logia* for "science, study, theory" [10].

Having that in mind, *being* in this particular case would be the enterprise and combined with study, results in the in deep study of an enterprise. Further details are made in the next paragraphs.

2.1.1 What is Enterprise Ontology?

Based on [11], Enterprise Ontology can be strictly defined as "a formal and

explicit specification of a shared conceptualization among a community of people of an enterprise (or part of it)*"*.

The full essence of the operation of an organization is what Enterprise Ontology is all about. This means that Enterprise Ontology is totally independent from the current realizations and implementation of the organization [12].

According to Dietz [11], the theory that underlies the notion of Enterprise Ontology is called the PSI-Theory. PSI-theory stands for Performance in Social Interaction. This theory is used by Dietz to construct a methodology which provides an ontological model of an organization, a model that should meet five quality requirements: the needed to be coherent, comprehensive, consistent, and concise and that only shows the essence of the operation of an enterprise model [11].

Based on [11], **coherent** it means that the distinguished aspect models constitute a logical and truly integral whole. By **comprehensive** it means that all relevant issues are covered. By **consistent** it means that all the aspect models are free from inconsistency. By **concise** it means that no superfluous issues are regarded in it, that the whole model is compact and succinct. And for the most important matter, is that this model should be essential, that only shows the **essence** of the enterprise, its deepen and its insight structure [11].

This referred methodology is called the Design and Engineering Methodology for Organizations, (DEMO) and is defined on Sub-chapter 2.2.

Why Enterprise Ontology?

The ontological model of an enterprise offers a reduction of complexity of over 90% [11]. This reduction of complexity is welcomed and turns any organization totally manageable and transparent for its manager. With that, it also shows the coherence between all the fields of the enterprise, such as business processes and the organization structure [12].

2.1.2 The PSI-Theory

The theory that supports the notion of Enterprise Ontology is the PSI-Theory and its ultimate goal is to capture the full essence of an organization. For that, four axioms were created [11].

Operation Axiom

The first axiom is called the **Operation Axiom** and states that Subjects in one organization normally perform two kinds of acts: production acts and coordination acts. The subject responsible for performing those acts is the actor and being responsible for that turns him the performer of a certain actor role [11].

By performing production-acts (P-acts for short) the subjects contribute for the

creation of goods and services for the organization. As a result for performing P-acts there is the creation of production-facts (P-facts for short) that could be something that has been manufactured (material) or something that has been done or decided (immaterial) [12].

By performing coordination-acts (C-acts for short) the subjects enter into and comply with the commitments regarding the production-acts. A C-act is performed by one actor, the performer, and directed to another actor, the addressee. C-acts are intentions (request, promise, question, etc). The result of a C-act is a coordination fact (C-fact for short) [12].

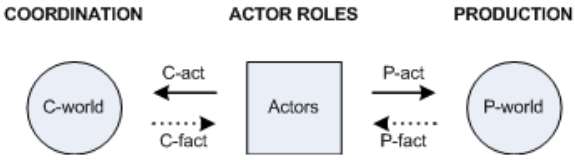


Figure 3: DEMO - Operation axiom

Source: Adapted from [11]

Transaction Axiom

The second axiom is the Transaction Axiom and states that both coordination and production acts are performed as steps in universal sociological patterns called transactions. The axiom shows that this patterns of coordination are the same for all type of organization. Figure 4 shows the transaction standard pattern.

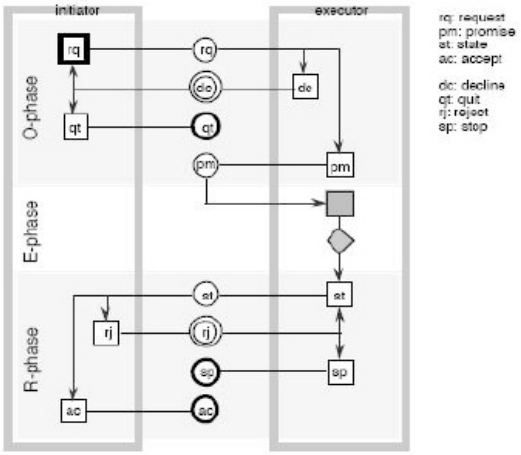


Figure 4: DEMO - Transaction standard pattern

Source:Adapted from [11]

In one transaction two actors are involved, the initiator and the executor. One transaction is composed by 3 phases. The Order-Phase (O-Phase) the Execution Phase (E-Phases) and the Result-Phase (R-Phase). In the O-Phase is where the initiator and the executor of the transaction tries to reach an

agreement about the intended result of the transaction. In the E-Phase is where the P-Fact agreed upon the O-Phase came to execution.

A transaction ends with the R-Phase, where the initiator and the executor work to reach an understanding about the P-Fact produced on the execution phase. The initiator can either reject or accept the result.

Composition Axiom

The composition axiom describes how the P-Facts are interrelated. It also states that every transaction is enclosed in some other transaction or it is a costumer transaction or it is a self activation transaction. According to Dietz this axiom is a well-founded definition for the notion of business process [11].

According to [11]:

> A business process is a collection of causally related transaction types, such that the starting step of either a request performed by an actor role in the environment (external activation) or a request by an internal actor role to itself (self-activation).

Distinction Axiom

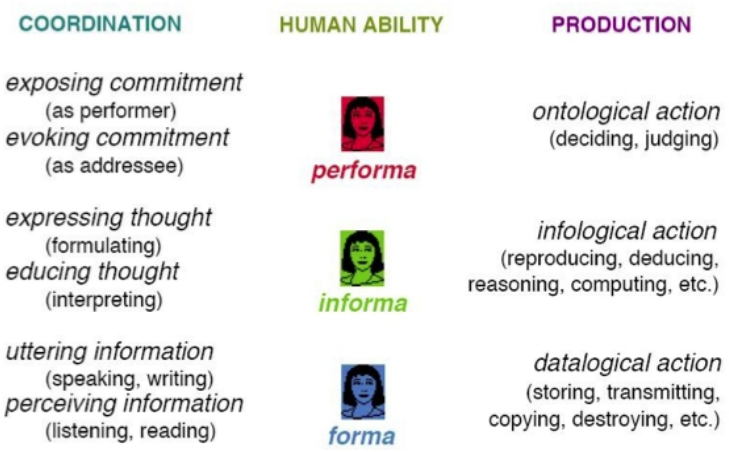


Figure 5: The PSI-Theory - Distinction axiom summary

Source: Adapted from [11]

The distinction axiom that can be seen on Figure 5 states that exists three basic human abilities performed by the transaction actors: performa, informa and forma. It is because of this axiom that a substantial reduction of complexity on both coordination and production of an organization is achieved [12].

The forma ability is related with the form aspects of communication and information [12].

The informa ability concerns the content aspects of communication and information [12].

The performa ability is about bringing out the new and original things, directly or indirectly by communication [12].

2.1.3 The Organization Theorem

The organization theorem - Figure 6 - combines the advantages of the four previous depicted axioms into one concise, comprehensive, coherent and consistent notion of an enterprise [12]. The organization theorem states that the organization of an enterprise is a heterogeneous system that is constituted as the layered integration of three homogeneous systems: B-organization

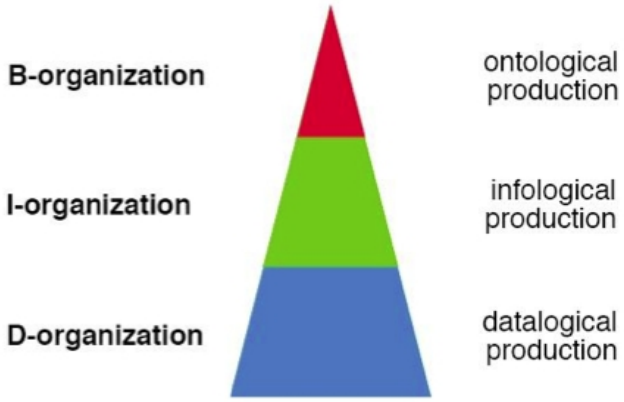


Figure 6: Organization theorem representation

Source: Adapted from [11]

(Business organization), the I-organization (Intellect) and the D-organization (Document) [12]. These three systems are called the aspect systems of the total organization [12] and can be seen on Figure 6.

2.1.4 Conclusion

This chapter was used to give a fully insight about enterprise ontology and its primary goals. In the PSI-Theory section, was described the four axioms of enterprise ontology and as stated its major goal, that is extract from each of the axioms, the full essence of an organization from its actual appearance.

The organization theorem is intended to combine the benefits of each of the four axioms into one concise, comprehensive, coherent and consistent notion of an enterprise.

With this, is fair to acknowledge that the enterprise ontology described by Dietz describes a very well-founded and defined theory about enterprises.

2.2 DEMO Methodology

The DEMO methodology is based on the PSI-Theory of Enterprise Ontology that was previously explained and describes a set of models and diagrams to model an organization.

According to [13] and after a ten year study executing 28 projects, DEMO has been declared as an excellent methodology for the (re)-design of organizations. With that, it became to be widely accepted in both scientific research and practical appliance [12]. Being more specific, the DEMO methodology provides a method to represent the full essence of an enterprise on an ontological level.

The essential information of the enterprise being modeled is visualized in a set

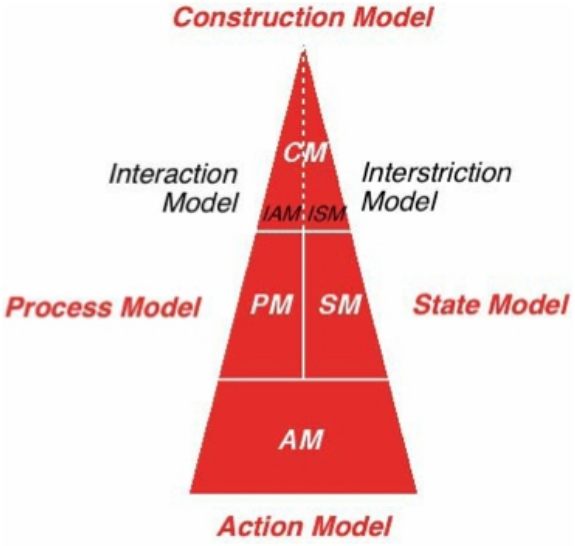


Figure 7: The ontological aspect models

Source: Adapted from [11]

of models and each model represent different aspects of the enterprise.

The referred models are: Construction Model (CM), Process Model (PM), Action Model (AM) and the State Model (SM) and they are constructed correlated with each other representing coherent information in a platform independent way [5].

The CM specifies the construction of the organization and its transaction types. It also states the actor roles and the transactions between them, the information links between the actor roles and the information banks [11]. Due to activeness and the passiveness of the influence between actor roles the CM is divided into the Interaction Model (IAM) and the Intersection Model (ISM). The IAM is where the active influences between actors are shown with the initiator and the executor. ISM contains the internal and external information banks and also

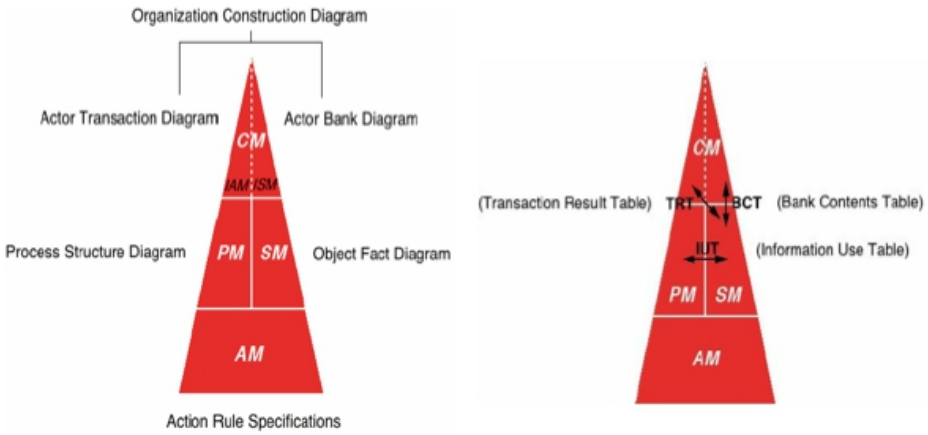


Figure 8: DEMO methodology: Diagrams and cross-model tables

Source:[11]

the information links between the actor roles and the information banks [11].

The PM contains, for every transaction in CM a specific transaction pattern and their causal and conditional relationships between transactions [11].

AM is the rule base that serves as guidelines for the actors actions. The action rules are composed by all the information from the previous models, CM, PM and SM, meaning that all the information about enterprises business is covered in this model, being one of the most important models on the methodology [11].

The State Model (SM) specifies the object classes, fact types, result types and the ontological coexistence rules [11].

All the previous depicted models constitute an essential representation of an enterprise in the business domain. With this, is fair to admit that DEMO is a well-defined methodology that is extremely useful on nowadays enterprise representation.

2.2.1 DEMO meta model

The DEMO meta model is in simple words the skeleton of the DEMO models and specifies the construction and the relationship between them (CM, PM, AM and SM) and can be seen on Figure 9. To represent the DEMO meta-model, is used the WOSL[4] (World Ontology Specification Language) specification language. The meta model is considered as a big State Model.

4 WOSL is specified in Sub-chapter 2.3-WOSL (World Ontology Specification Language) and further information is found on Chapter 5 of [8].

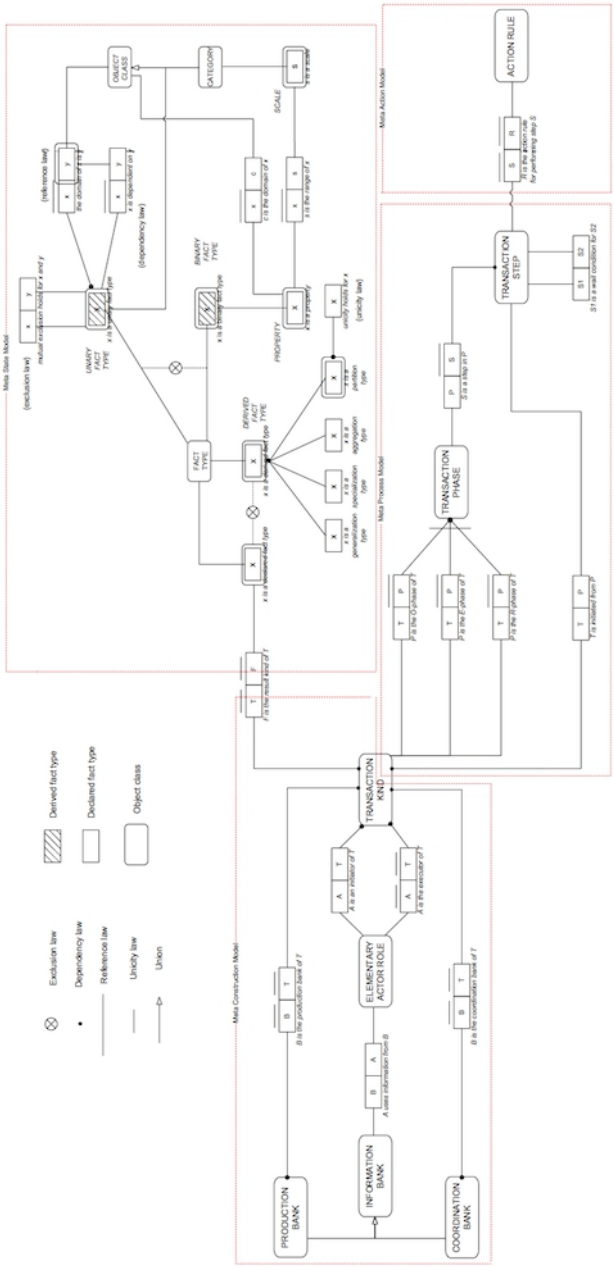


Figure 9: DEMO Meta-model

Source: [11]

2.3 WOSL (World Ontology Specification Language)

WOSL is a specification language presented by Jean L.G Dietz and is used to specify world ontologies. A world ontology is based on the specification of the state space and the transition space of that specific world. The state space

means the set of allowed or lawful states. It is specified by means of the state base and the existence laws. The set of statum types of which instances can exist in a state of the world are specified by the state base. The inclusion or exclusion of the coexistence of stata are determined by the existence laws. The transition space means the set of allowed or lawful sequences of transitions. It is specified by the transition base and the occurrence laws. The set of factum types of which instances may occur in the world are specified by the transition base. Each instance has a time stamp which is the event time.

The WOSL specification language is important for this work since it is the language used to specify the UEAOM[5] (Universal Enterprise Adaptive Object Model) and some of the models used on the DEMO methodology.

2.3.1 Stata and Facta

A world can be at any moment in a particular state, which is defined as a set of objects. These, are said to be current objects during the time that the state at that particular time prevails. A state change is called a transition. The occurrence of a transition is called an event. By consequence, a transition can happen several times during the lifetime of a specific world, instead of events that are unique. An event is caused by an act. In order to understand profoundly what a state of a world is, and what a state transition is, it is necessary to distinguish between two kinds of objects, which we are called stata (singular: statum) and facta (singular: factum).

A statum is something that is just the case and that will always be the case; it is constant. Example: The author of book title T is A. The existence of this particular objects are timeless. A derived statum is defined by its derivation rule. The being specified of this rule is the only necessary and sufficient condition for the existence of the derived statum. This marks an important difference between a world and a database system about that world. E.g. the age of a person in some world exists at any moment, however, it has to be computed when it is needed. Stata are subject to existence laws. These laws require or prohibit the coexistence of stata. For example, if the author of some book is “Ludwig Wittgenstein”, it cannot also be “John Irving”.

A factum is the result or the effect of an act. Example: book title T has been published. The becoming existent of a factum is a transition. Before the occurrence of the transition, it did not exist and after the occurrence it does exist. Facta are subject to occurrence laws. These laws allow or prohibit sequences of transitions. For example, sometime after the creation of the factum “loan L has been started”, the transition “loan L has been ended” might occur, and in between several other facta may have been created, like “the fine for loan L has been paid”.

5 Explained in Sub-chapter 2.6-UEAOM

2.3.2 WOSL Grammar

In order to keep the specification of the grammar of WOSL orderly and concise, it is presented in a number of figures.

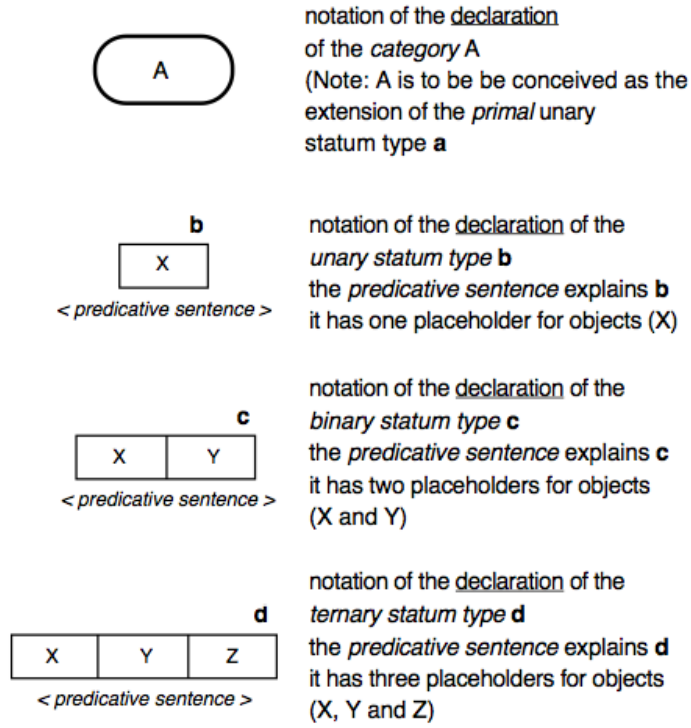


Figure 10: WOSL grammar - Statum type declaration (Part 1)

Source: Adapted from [11]

Figures 10 and 11 depicts the way in which statum types can be specified. By the declaration of a statum type is understood stating that the statum type belongs to the state base of the world under consideration. Statum types can be declared intensionally or extensionally. By intensional we mean the notation of the statum type as a unary, binary, ternary etc. concept type. Intensional notations are referred to be a bold small letter (or a string of small letters). Extensional notations are referred to by a capital letter (or a string of capital letters). To understand what a state of a world is, it is necessary to distinguish between two kinds of objects: stata and facta. WOSL language has several graphical pictures to represent these stata and facta.

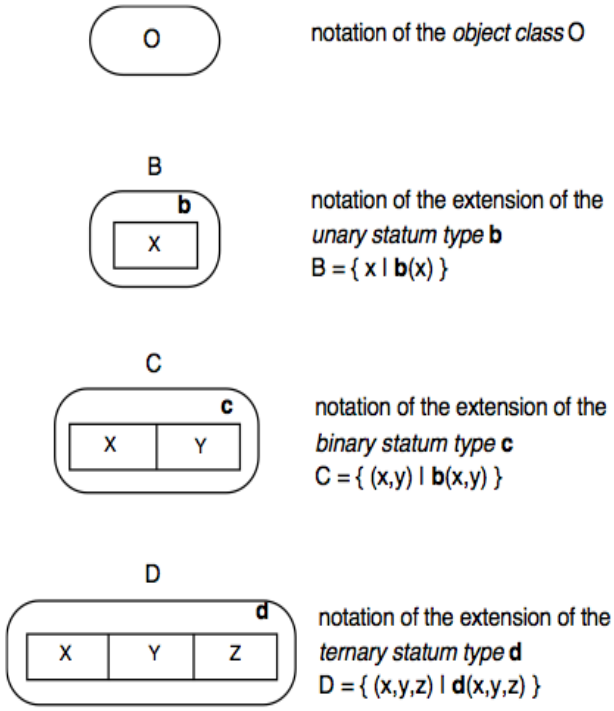


Figure 11: WOSL grammar - Statum type declaration (Part 2)

Source: Adapted from [11]

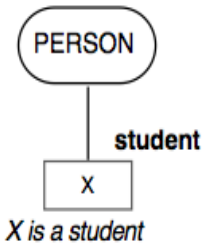


Figure 12: WOSL grammar - Reference law

Source: Adapted from [11]

Figure 12 shows an example of a reference law. Figure 13 shows a dependency law.

Figure 14 shows an unicity law.

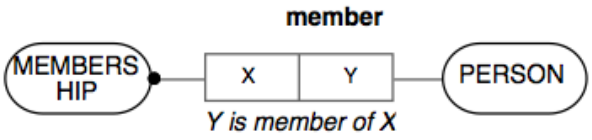


Figure 13: WOSL grammar - Dependency law

Source: Adapted from [11]

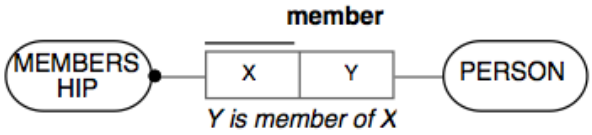


Figure 14: WOSL grammar - Unicity law

Source: Adapted from [11]

Figure 15 shows an example of a factum type.

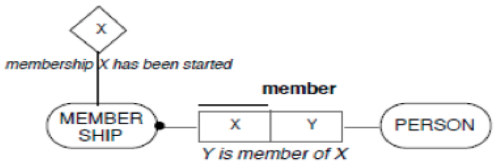


Figure 15: WOSL grammar - Factum type

Source: Adapted from [11]

2.4 BPM (Business Process Management)

In the business world domain, BPM acronym can be easily confused with other different expansions that are somehow related, as Business Process Modeling, Business Process Model and so on. Despite that, the most generic is BPM for Business Process Management and is the one that is focused in this chapter definition.

Business Process Management is, as said above, the most generic definition and refers to all the activities used to control the enterprise management. This chapter is aim to describe BPM and its most important related activities.

2.4.1 Introduction to BPM

What is Business Process Management?

After so many years having in mind that BPM (Business Process Management) exists there is not yet a most correct and true answer to this question. It seemed important to clarify this because there is not a universal truth about it. For this thesis was adopted the definition that seemed the most appropriated and reasonable for the sake of this project according some of the most quoted publications regarding that matter.

Based on [14] Business Process Management is the discipline that describes structured methods and techniques used to make a business process more efficient adaptive and effective for accomplishing a specific task within an organization. BPM techniques and methods also permits the identification and modification of existing processes in order to align them to future possibilities of change.

BPM could also be seen as the responsible for narrowing down the gap between the line of business and the IT department. According to a recent research based on [14], has been concluded that cooperation and coordination between this two normally separated departments, is quite important and very useful for a more valid and complete BPM.

Keep in mind that BPM is not a tool of software or even related with technology. However, it can involve technology and if used in the right circumstances and with valid justification could be a useful help to achieve efficiency on process-modeling organizations, which is what BPM is intended for. For instance, the use of process-modeling tools are extremely important since it will be difficult to complete complex process-modeling improvements without such tools [15].

Other word of precaution for organizations is they should be aware that BPM tools itself are just a piece of software and can do nothing related with business process improvements without the knowledge and the right methodologies.

Summing up and based on [15] that suggests that BPM is:

> The achievement of an organization's objectives through the improvement, management and control of essential business processes.

Table 1:BPM Definition terms, shows its details:

Table 1: BPM - Definition terms

Source: Adapted from [2]

Achievement	Realizing the strategic objectives as outlined in the organization’s strategic plan. At a project level, it is

	about realizing the value or business benefits as outlined in the project business case.
Organization	The organization in this context refers to an enterprise or parts of an enterprise, perhaps a business unit that is discrete in its own right. It is the end-to-end business processes associated with this part of an organization. This end-to-end focus will ensure that a silo approach does not develop.
Objectives	The objectives of a BPM implementation range from the strategic goals of the organization through to the individual process goals. It is about achieving the business outcomes or objectives. BPM is not an objective in itself, but rather a means to achieving an objective. It is not a solution looking for a problem.
Improvement	Improvement is about making the business processes more efficient and effective.
Management	Management refers to the process and people performance measurement and management. It is about organizing all the essential components and subcomponents for the processes. Arranging the people, their skills, motivation, performance measures, rewards, the processes themselves and the structure and systems necessary to support a process.
Control	BPM is about managing the end-to-end business processes and involves the full cycle of plan–do–check–act [16]. An essential component of control is to have the ability to measure correctly. If something cannot be measured, it can't be controlled and managed.
Essential	Not every process in an organization contributes towards the achievement of the organization’s strategic objectives. Essential processes are the ones that do.
Business	An implementation of BPM must have an impact on the business by delivering benefits. It should focus on the core business processes that are essential to the primary business activity – those processes that contribute towards the achievement of the strategic objectives of the organization.
Processes	What is a process? There are as many definitions of process as there are processes. Roger Burlton’s says

	that: “a true process comprises all the things we do to provide someone who cares with what they expect to receive.” [17] This covers a true end-to-end process, from the original trigger for the process to the ultimate stakeholder satisfaction. Burlton adds that the “ final test of a process’s completeness is whether the process delivers a clear product or service to an external stakeholder or another internal process”.

2.4.2 History

As everything in computer technologies related, BPM had ups and downs and a really tough road until it became the “next big thing” in business process related [15]. It stood out and continued from then, thanks to others various failed attempts for achieving the process-based organizational efficiency.

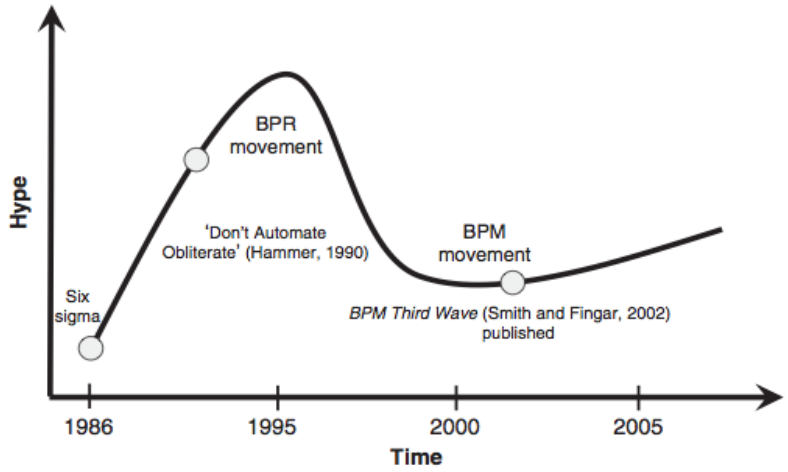


Figure 16: BPM hype cycle

Source: Adapted from [15]

Adapted from [15], Figure 16: BPM hype cycle, shows the last two decades of how the process cycle has progressed.

2.4.3 BPM life cycle

As seen above, BPM consists on techniques and methodologies of optimizing business processes profitability and efficiency. For that, there are many possibly approaches depending on who is implementing it or even what kind of result is expected for the process.

On this thesis was used and studied the simplest, however it is a very complete approach, so it is easily understood with all the necessary details.

The worth knowing approach is based on [18] and consists in five major phases: Model, Automate, Manage/Execute, Monitor and Optimize. In the next few paragraphs all of this phases are described in further detail.

Model

Such as much everything that is based on various phases - computer related - it must have a primary model, like a skeleton that is wrapped and perfected on the following phases. The first step is simple but at the same time extremely important. Is based on this model that the others phases rely on.

In this phase a high-level diagram of the process is created with the goal of gathering just enough information to understand conceptually how the process works, and which are the steps involved in it, without being mislead by its implementation details [18].

Automate

During this phase and based on the previous, the model is expanded in order to create a specific set of instructions and rules needed to run the process. At this point all the decisions regarding the details of how the implementation should be made, are done [18].

Execute/Manage

Execute phase is within the manage phase since it is totally related. It is the way that the manage of the process is made, by executing it.

The execution phase is basically to interpret the instructions created on the Automate phase, to manage the flow of the process since its beginning until its completion. After that and with the BPM software tool the workflow engine is responsible for creating tasks and automatically directing them to the right people or systems based on the process rules [18].

Monitor

The monitor phase is where process performance is measured, tracked and reviewed for potential and possible improvements [18].

Optimize

Optimize phase is used by managers to - regarding all the information learned from the manage phase - optimize the process. Things as enhancing the data collection forms, adding or removing tasks, automating tasks that were made manually are made in this phase. The ultimate goal of this phase is to identify changes that will certainly improve the all process [18].

2.4.4 Conclusion

After this simple and summed up insight about BPM and what it could do for

process modeling on organizations, is easy to understand that it has lots of benefits. It is not a software tool that solve all issues process related but will certainly help if used with knowledge and with the right amount of technology.

Worth saying that, and maybe one of the most ignored aspect of BPM, is its continuous improvement. As an organization change, with it processes and environment changes, along with technologies. Also with that, processes need to change in order to meet the actual needs.

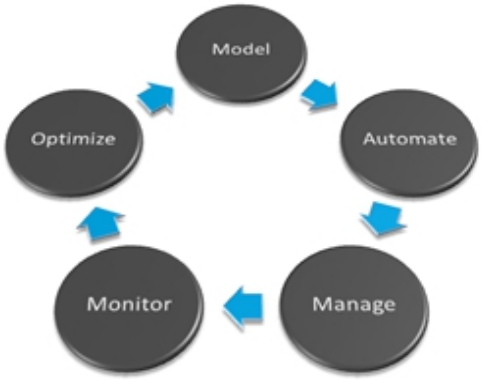


Figure 17: BPM continuous life cycle

Source: [18]

The five steps of business process management can be considered as part of a continuous life cycle. Once a potentially change is identified, the cycle of implementation begin all over again and again until the changes are successfully made. Having all this in mind the organization willing to implement BPM methodologies and techniques makes a continual and incremental improvement to its business processes [18].

2.5 BPMN (Business Process Model and Notation)

BPMN, stands for Business Process Model and Notation. What is BPMN, its history of development and its actual notation is what is shown in the next sub-chapters.

2.5.1 Introduction to BPMN

What is it?

BPMN is basically a method to graphically represent the steps of a business process in a business process model in a form of diagram. However being different in context it is very similar to a flowchart. The notation of BPMN was specifically designed for coordinate the sequence of the processes and the

way that messages flow between activities, processes and participants.

Flowchart vs Modeling

Even being very similar there are some differences that deserves some concerning.

A flowchart, and based in [19], is "A graphical representation of the sequence of activities, steps, and decision points that occur in a particular, discrete process". A flowchart is mainly used to explain the sequence of a process graphically, to improve communication and obtain business-user validation, to identify bottlenecks and loops, to assist with problem analysis, to provide a blueprint to development and to identify the variations in process activity [19]. The modeling process extends the flowchart for things that cannot be enumerated on the basic flowchart: as mapping dependencies and related flows, adding data intelligence to the steps, enabling simulation of flows to check for efficiencies and bottlenecks, enabling reuse of mapped chart elements and finally supporting future monitoring of improved processes.

Why it is important?

Over the past few years the world of business processes for organizations has dramatically changed. The organizations demanding for more complex processes has increased and with that a necessary reformulation of how the business processes should be made was necessary. Even a medium sized organization that happen to use the IT's to its normal daily work makes use of coordination between participants and that could became quite complex if not treated well enough. Until then and without BPMN there wasn't a standard modeling technique to avoid this kind of complexity so that BPMN has been developed to provide users with a global and free notation.

Development history

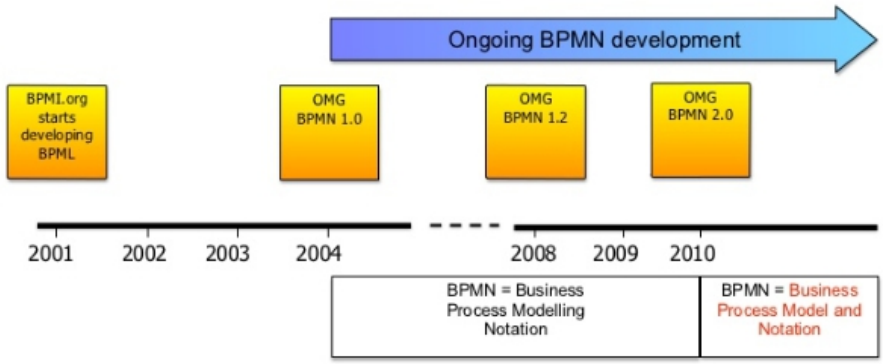


Figure 18: BPMN Development history

Source:[21]

Bringing up some history, all began on the beginning of 20th century, when the

BPMI (Business Process Management Initiative) started to develop the BPML, which stands for Business Process Management Language. BPML is an Extensible Markup Language as a mean of modeling business processes. Along with BPML, appeared the BPQL (Business Process Query Language), developed by Initiative members as a standard management interface for deployment and execution of defined business processes. Both BPMN and BPQL were open specifications and the first draft of BPML was submitted in August 2000 and made available in March 2001. Was BPMI intention to proceed on the next few years the development of both specifications.

However in 2004, BPMI merged with OMG (Object Management Group) and since then both worked on the creation of BPMN. The actual version of BPMN is the version launched in August 2009 as BPMN 2.0 Beta 1, today it stills by BPMN version 2.0.

OMG is an international non-profit organization and open membership, that was founded by a partnership between some well-known companies back then, in the late's 1980, more precisely in the year 1989. With a motto of "We deliver the standard" is quite straightforward what they intended to do with the creation of such organization. Their focus was about the modeling of programs, systems and businesses processes standardization. Since they only provide specification, all the implementation should be based on the same specification, so all the business related implementations could be made having the same guidelines leading to a final similar, understandable implementation [20].

Main goal

The main goal of BPMN is essentially to support BPM (Business Process Management) by providing the ability to both technical and business users to easily understand the provided notation. It could also represent extremely complex process semantics and even still be quite easy to understand.

2.5.2 Structure

BPMN gives the ability to model three different, but very similar, aspects of any business process. The Process itself, the Collaboration, and the Choreography. This three different aspects of any organization can be represented on different types of diagrams.

The process is seen as a sequence of activities and events related to a business process.

The collaboration is basically a process that has two or more participants that exchange messages between each other. The major importance of a collaboration diagram is essentially the sequencing of the activities the events and the messages that are traded between the participants [21].

Choreography is a sequence of interactions between participants [21]. This

type of representation was only introduced on BPMN version 2.0 and is still not used very often.

2.5.3 Basic Notation 2.0

Based on [21], a business process is defined as simple as: “A sequence of activities performed by one or more business participants in order to deliver value to he business”.

BPMN models have their own notation composed by a set of graphical elements used to construct its own diagrams. Essentially, a process is defined by a set of elements that compose different types of flow nodes that could be connected by sequence flows, in order to form a kind of flowchart [21].

All the notation described in the next paragraphs are based on the BPMN version 2.0.

The process diagram

BPMN have four simple basic elements categories for the construction of its diagrams [21]:

- **Flow Objects** are composed by events, activities and gateways.
  - Events is what happens during the process;
  - Activities are worked performed in the process;
  - Gateways control the flow along the the process.
- **Connecting Objects** are composed by sequence flow, message flow and associations.
  - Sequence flows are according the flow of the activities;
  - Message flows are messages between process participants;
  - Associations associate text or data to the modeling elements.
- **Swim lanes** are composed by pools and lanes.
  - Pools represent a participant in the process;
  - Lanes represent a group of related activities.
- **Artifacts** are composed by data objects, groups and annotations.
  - Data objects show to the user which data is required or produced in one specific activity;
  - Group is simply used to group related but different activities so it became more understandable for the context in which is related but does not affect in any way the flow of the diagram;
  - Annotation is used to give the user an understandable impression about something.

Figures 19, 20, 21, 22 and 23 illustrates some examples of BPMN

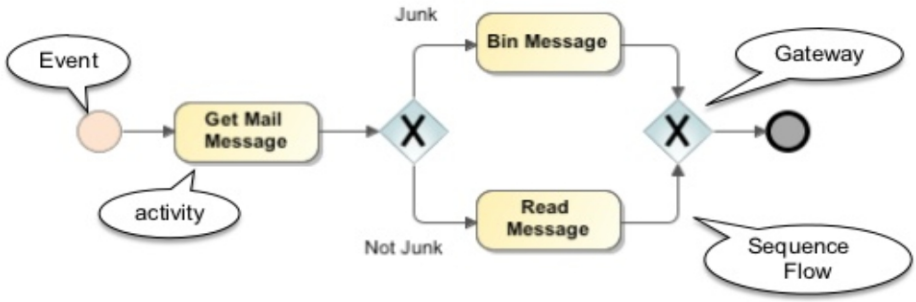


Figure 19: Basic BPMN diagram - Flow objects

Source:[21]

constructions and notation used to construct its respective processes.

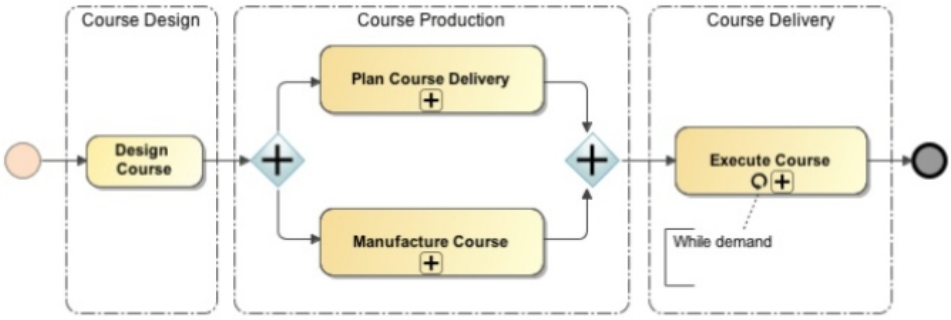


Figure 20: Basic BPMN diagram - Using artifacts

Source:[21]

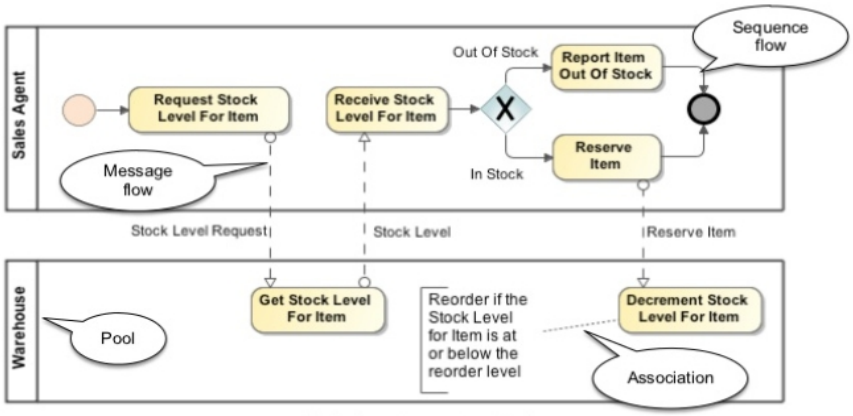


Figure 21: Basic BPMN diagram - Connecting objects

Source:[21]

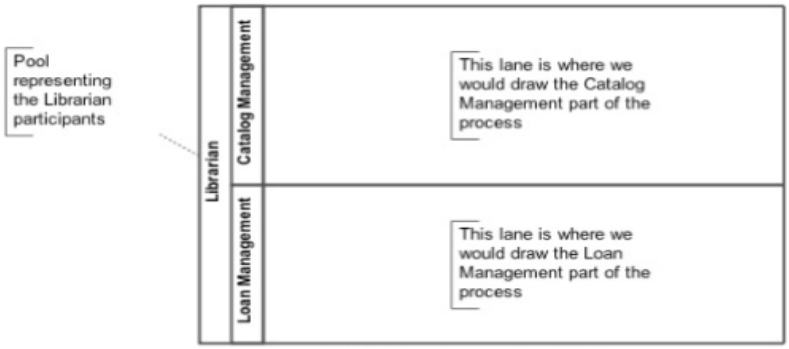


Figure 22: Basic BPMN diagram - Collaboration diagram

Source:[21]

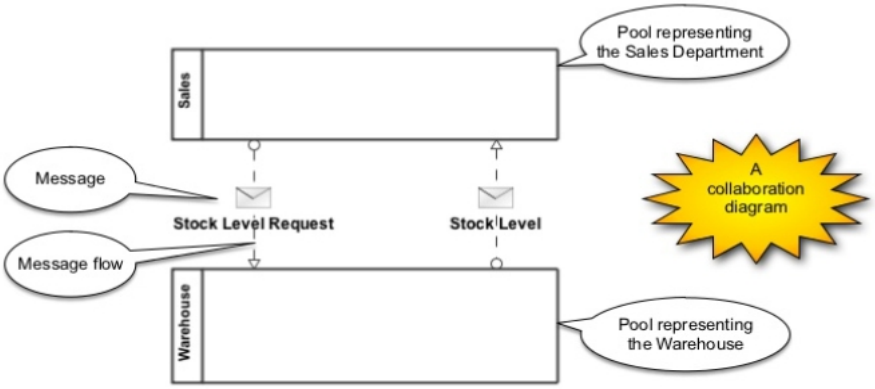


Figure 23: Basic BPMN diagram - Collaboration diagram

Source:[21]

Process Semantics

The so called token game is basically an imaginary focus to control the flowing of the process. The token runs the entire process via the sequence flow of the diagram [21].

Figure 24: BPMN Process Semantic - The token game

Source:[21]

Process instances

One process may have one or more instances of its implementation. This means that could exist for the same process diagram various running instances at the same time.

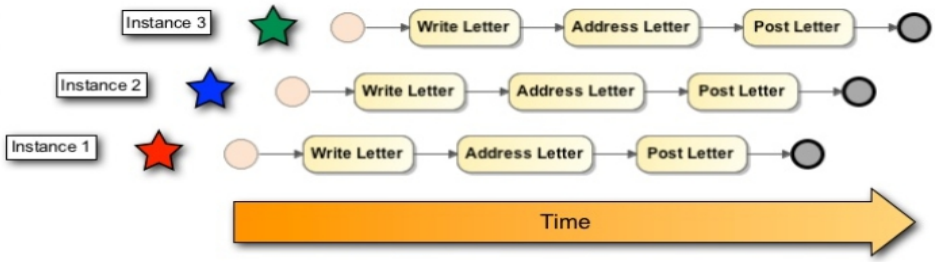


Figure 25: BPMN Process instances

Source:[21]

Events

An event in a BPMN process is basically something that happens during the process .They normally affect the flow of the process and usually have a cause (trigger) or an impact (result). Is something that could be externally or internally triggered or simply an achievement of something noticeable of interest for the process [22].

There are four categories for events and they are graphically distinguished by its boundary style, according to the next figure based on [21]:

Syntax	Boundary	Name	Semantics
	Thin	Start events	Begin the process by emitting a token
	Double	Intermediate events	Occur during the process
		Intermediate boundary events	Intermediate events that are attached to an activity boundary
	Thick	End events	Terminate a process flow by consuming a token

Figure 26: BPMN 2.0 notation - Events categories

Source:Adapted from [21]

Inside of each category and adapted from [23], all the types of events available in the BPMN 2.0 notation are showed and described on Figure 27:

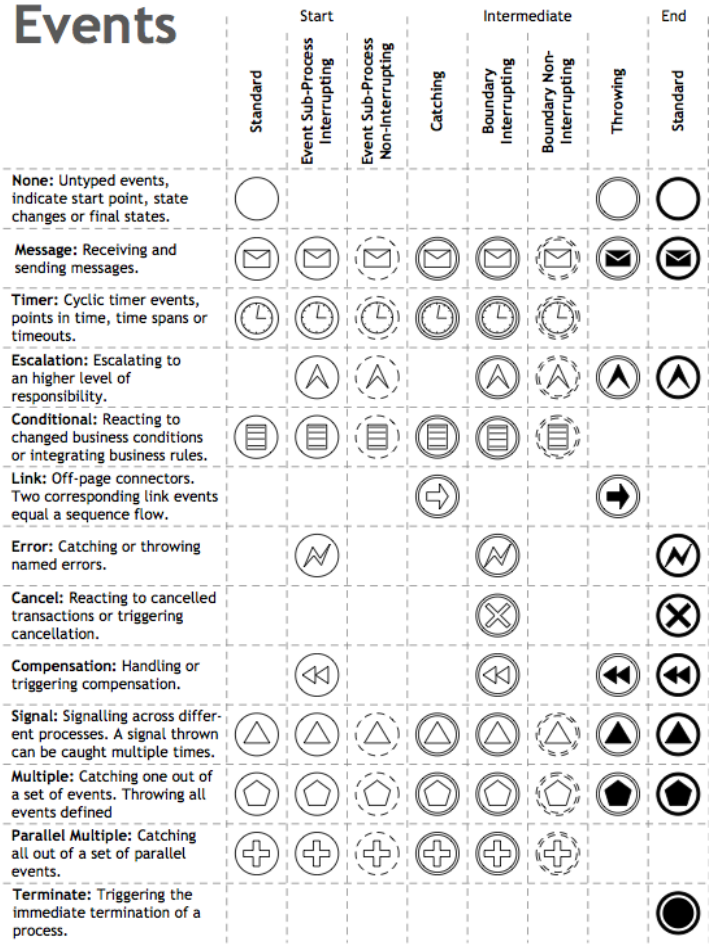


Figure 27: BPMN 2.0 notation - Types of events

Source: Adapted from [23]

Activities

Activities as has been previously exposed, is work performed within the process. Adapted from [23], and in Figure 28, there are the activities explained:

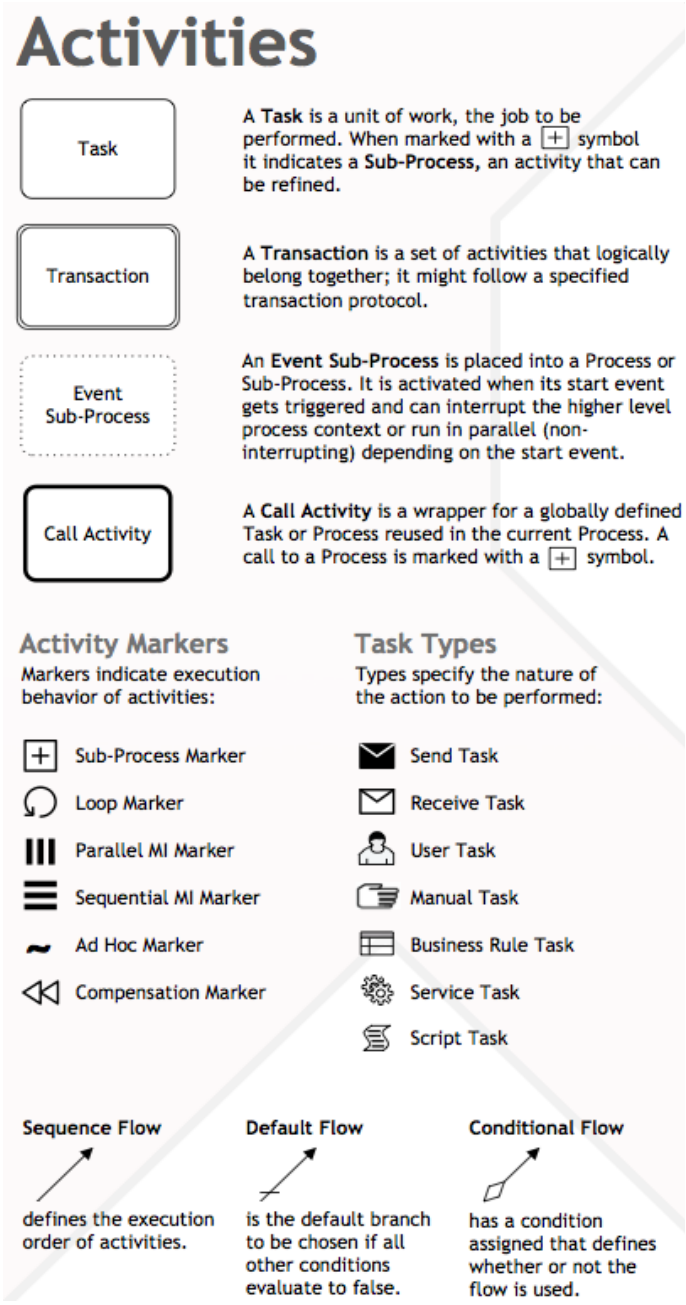


Figure 28: BPMN 2.0 notation - Type of activities

Source: Adapted from [23]

Gateways

Gateways are responsible for directing the paths of the diagram, depending on the conditions expressed. Based on [23] gateways are described as follows:

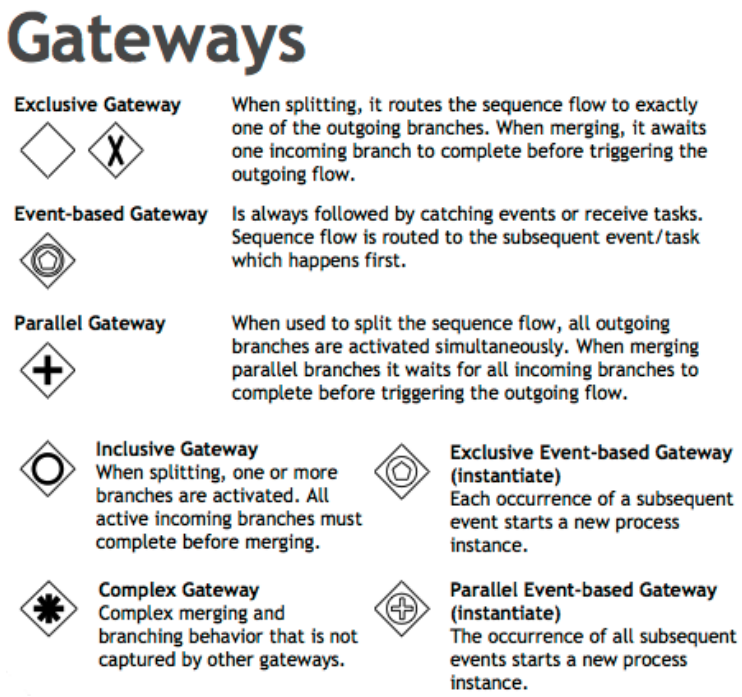


Figure 29: BPMN 2.0 notation - Gateways

Source: Adapted from [23]

Swim-lanes

Swim-lanes are a visual way of grouping and categorizing related activities. Adapted from [23], there are the Swim-lanes expressed on Figure 30:

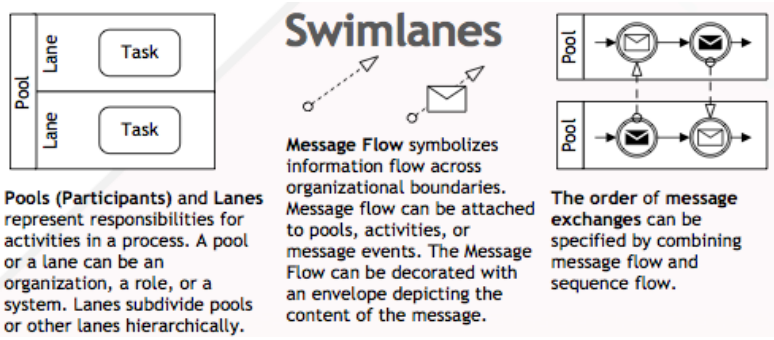


Figure 30: BPMN 2.0 notation - Swim-lanes

Source:Adapted from [23]

Full process diagrams examples

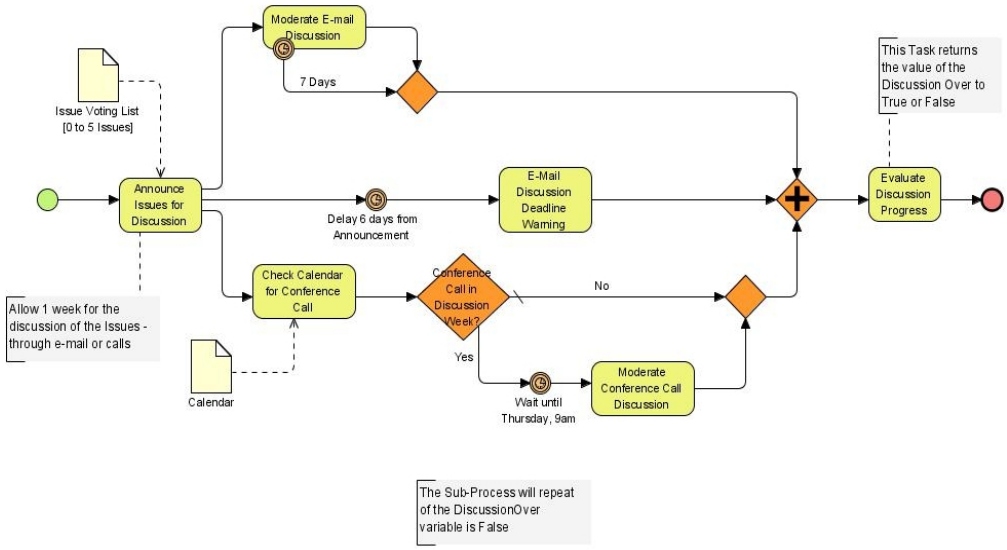


Figure 31: BPMN notation - Process diagram - example 1

Source: Adapted from [37]

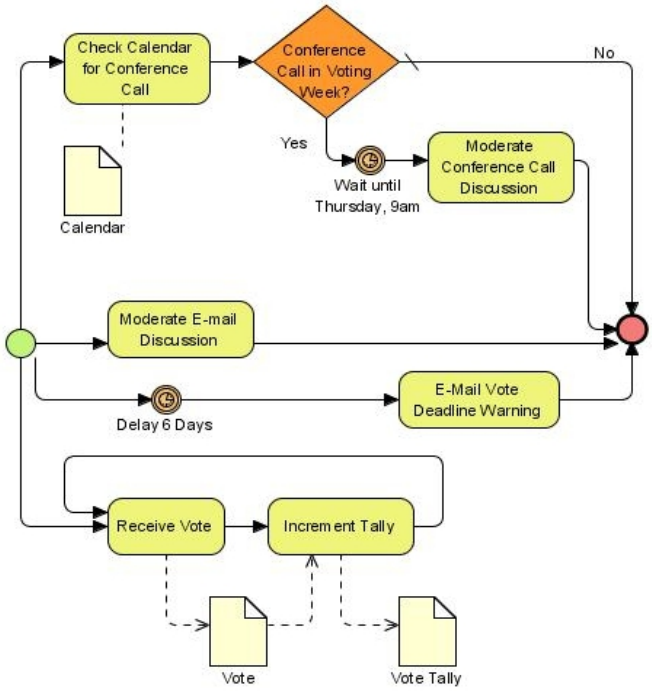


Figure 32: BPMN notation - Process diagram - example 2

Source: Adapted from [37]

2.5.4 Conclusion

BPMN is seen as a the standard notation for Business Process Management as it is extremely expandable. It is based on a flowcharting technique, very similar to activity diagrams from unified modeling language. Using BPMN as a standard for capturing and representing business processes is definitely valuable and helpful for any organization however it is a complex way of doing it. The wide range of options for its construction and even the possibility of mixing constructions allows the creation of models with semantic errors [24]. These type of errors in the early phases of systems development are normally the most serious and the most difficult to find and solve so it must be avoided. There is not a pattern or solid semantics for the construction of BPMN diagrams. Regarding this situation and for the sake of this project, rules and guidelines for the BMPN construction are suggested.

2.6 UEAOM

UEAOM stands for Universal Enterprise Adaptive Object Model. Despite being a recent proposal, some work developed on this thesis is based in this proposal. Its advantages and main capabilities available are also described and acknowledged.

Along with the UEAOM, it is important to give an idea about its major foundations, the Adaptive Object Model pattern.

2.6.1 Introduction

The Universal Enterprise Adaptive Object Model (UEAOM) is a conceptual schema being used as the base for the effort to create a wiki-based system that allow an effective enterprise modeling, language independently, so that all the organization changes can be made, saved and propagated in a runtime environment.

For starters, an AOM (Adaptive Object Model) is a common architectural style for systems in which classes, attributes, relationships and behaviors of applications are represented as metadata, allowing them to change in a runtime environment [25].

An AOM system provides great flexibility for applications, allowing relationships, attributes and behaviors to be changed at runtime by programmers, and sometimes by the final users. These systems can be easily adapted on environments where business rules are in constant changing [25].

It is a model based on instances rather than classes. Users change the meta-data (object model) to reflect changes in the domain. These changes modify the system's behavior. In other words, it stores its Object-Model in a database

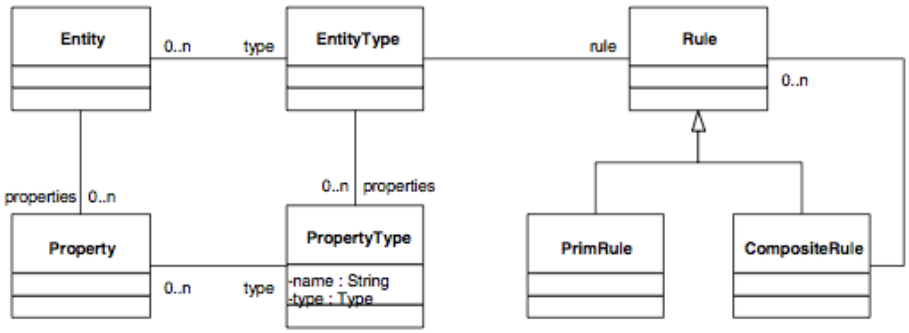


Figure 33: TypeSquare with rules

Source: Adapted from [26]

and interprets it. Consequently, the object model is active, when you change it, the system changes immediately [26].

Based on its presentation proposal [27], UEAOM is:

> A novel conceptual model that systematizes the integrated management and adaptation of: (1) enterprise models, (2) their representations, (3) their underlying meta-models, i.e., their abstract syntax and (4) the representation rules, i.e., concrete syntax for the respective models. All this for different modeling languages and also different versions of these languages. Thanks to our original use of the adaptive object model and type square patterns – normally applied in the context of software engineering, but here applied for enterprise engineering – we manage to provide a strong conceptual foundation for the development of software tools that will allow a precise and coherent specification of models and their evolution and also of meta-models and their evolution.

2.6.2 UEAOM model

The long term goal of this proposal is the development of a wiki-based system that provides an effective integrated enterprise modeling, allowing a dynamic evolution of meta-models, models and their representations along with an

intuitive navigation through their elements and their semantics. Also permits wide-spread model interpretation, distributed model creation and change, reflecting enterprise changes [27].

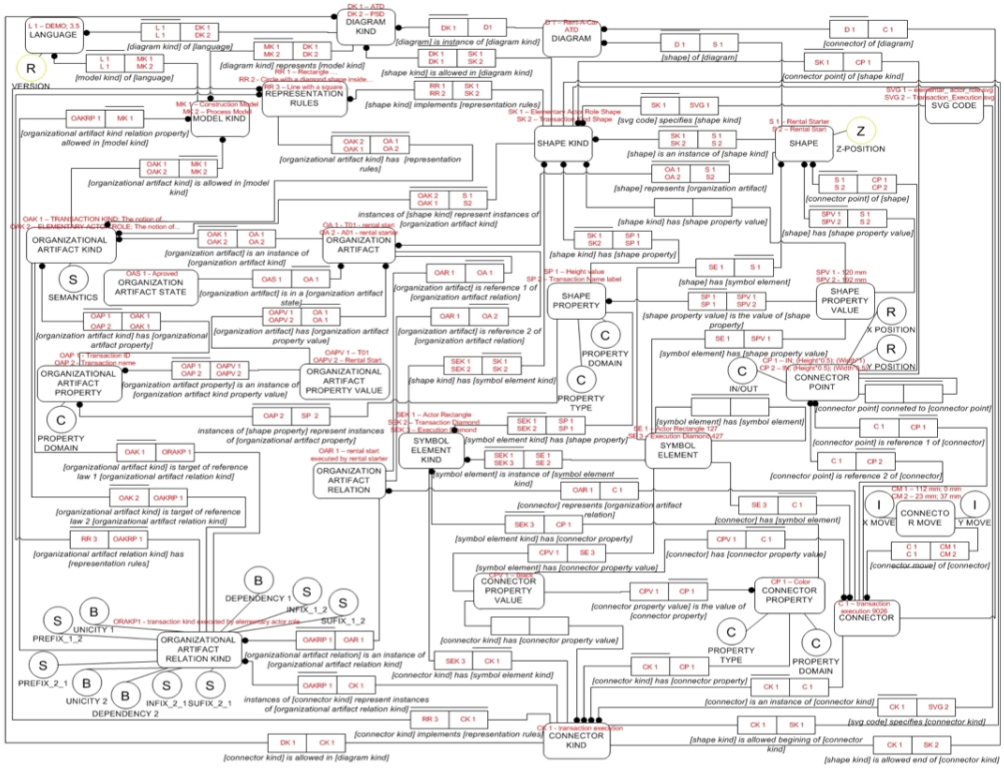


Figure 34: Universal Enterprise Adaptive Object Model

Source: From [27]

From this thinking and based on AOM as described previously, rises the so called UEAOM that is shown in Figure 34.

The AOM pattern is applied so that each page of the semantic property of the semantic wiki-based system corresponds to instances of the AOM.

Wiki pages, that are instances of class DIAGRAM, automatically generate SVG diagrams based on shape and connector pages. These pages also allow dynamic editing of diagrams and underlying models. The type-square pattern [26] - shown in Figure 33 - is applied four times as to allow run-time dynamic change of: (1) meta-model elements, (2) model elements, (3) shape elements and (4) connector elements. [27] The UEAOM is represented with the World Ontology Specification Language[6] (WOSL). WOSL is based in Object Role Modeling language.[28] In [29] a relation between Adaptive Object Model pattern and the MOF standard is presented, where run-time instances of the operational level are equivalent to MOF's M0 and knowledge level; classes, attributes, relations and behavior is equivalent to M1, being M2 an equivalent to the models used to define an AOM. In UEAOM all these MOF levels are projected as run-time instances.

In this prototype system, both organization artifacts – i.e., concrete

6 The WOSL is described on Sub-chapter 2.3-WOSL (World Ontology Specification Language)

organization models – and organization artifact kinds – are taken as instances, i.e., the meta-model specification or, in other words, the abstract syntax. So both M1 and M2 levels of the MOF framework exist and change at run-time. The MOF is too software development oriented and too complex for this needs. The main contribution of this proposal is to apply these fundamental theoretical foundations and adapt them to the field of enterprise ontology [27].

After the UEAOM contextualization a simple explanation of its content is due.

The UEAOM's classes are not explicit specifying syntaxes of a particular modeling language instead and while instantiating these classes, is to specify any syntax of any modeling language, along with particular models of each language, and also their evolution, all this in run-time environment [27].

2.6.3 Abstract syntax

Relevant classes for the specification of the abstract syntax of any version of any language are presented in Figure 35.

The main concepts of the abstract syntax specification are expressed in the classes LANGUAGE, MODEL KIND, ORGANIZATIONAL ARTIFACT KIND (OAK) and ORGANIZATIONAL ARTIFACT RELATION KIND (OARK). They specify all allowed artifacts (e.g. transaction kind OAK and

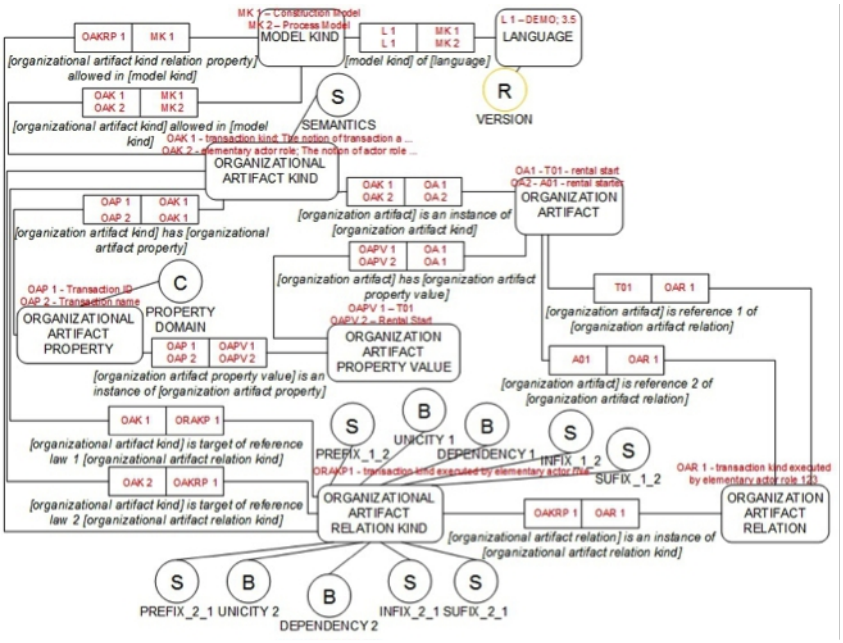


Figure 35: UEAOM - Abstract syntax

Source: Adapted from [27]

transaction execution relation OARK) for different types of models that can exist for different languages. Class ORGANIZATIONAL ARTIFACT RELATION KIND has ten properties that can be divided in two groups of five

where each group specifies one of the two sides of an allowed relation between two OAKs [27]. The ones named prefix, infix and suffix specify the formulation that can be done around the names of the two OAKs being related. Most of the times, only the infix needs to be specified. With the unicity and dependency properties the cardinality of the relation is specified and which OAKs are mandatory or not to participate in the relation.

Reference law fact types specify which two OAKs are allowed to participate in this relation. Practical example of the first set of the referred 5 properties: F *Transaction Kind* T ***is initiated by*** *Elementary Actor Role* corresponds to a set of Dependency 1, Reference law 1, Unicitiy 1, Infix_1_2 and Reference law 2. F *Elementary Actor Role* T ***is initiator of*** *Transaction Kind* would be its corresponding Dependency 2, Reference law 2, Unicitiy 2, Infix_2_1 and Reference law 1. Thanks to this part of our UEAOM specification we allow a precise and formal formulation of the abstract syntax of models, already giving considerable semantics thanks to the prefix, infix, suffix and OAK names that can be composed in formulations for each direction of the relation. Instances of class ORGANIZATION ARTIFACT PROPERTY specify intrinsic properties of OAKs, like identifiers and names. The respective property PROPERTY DOMAIN allows us to specify the domain for each intrinsic property of an OAK (e.g., string, number, etc.). Examples of instances are property *transaction id* with domain *T<number>* or *transaction name* with domain *<string>* [27].

2.6.4 Concrete syntax

The UEAOM classes that allow the specification of rules for the concrete representation of models, i.e., the concrete syntax, are presented next. These classes, together with all their inter-relating fact types are present in Figure 36. With the class SHAPE KIND, instances of the types of shapes allowed to be part of diagram kinds representing certain model kinds are specified. These shape kinds are also specifically connected to the OAKs whose instances they will represent. For instance, the *elementary actor role shape* is allowed in diagram kind *Actor Transaction Diagram*, that represents the *construction model* of DEMO language. Instances of this shape represent instances of OAK *actor role*.

With SHAPE PROPERTY, is specified the properties for each shape, e.g., *line color* and *actor id label* of actor role shape. Instances of CONNECTOR KIND specify allowed representations for OAKRs, e.g. *transaction initiation connector* instances represent instances of OAKR *transaction initiation*. With CONNECTOR PROPERTY, the properties of each connector are specified, e.g., for the just mentioned connector, *line color: black* and *line dashing: continuous*. Instances of REPRESENTATION RULES, class are an informal textual based specification of rules on how ORGANIZATIONAL ARTIFACT KINDS and ORGANIZATIONAL ARTIFACT RELATION KINDS should be represented. These rules are taken in consideration in either SHAPES or

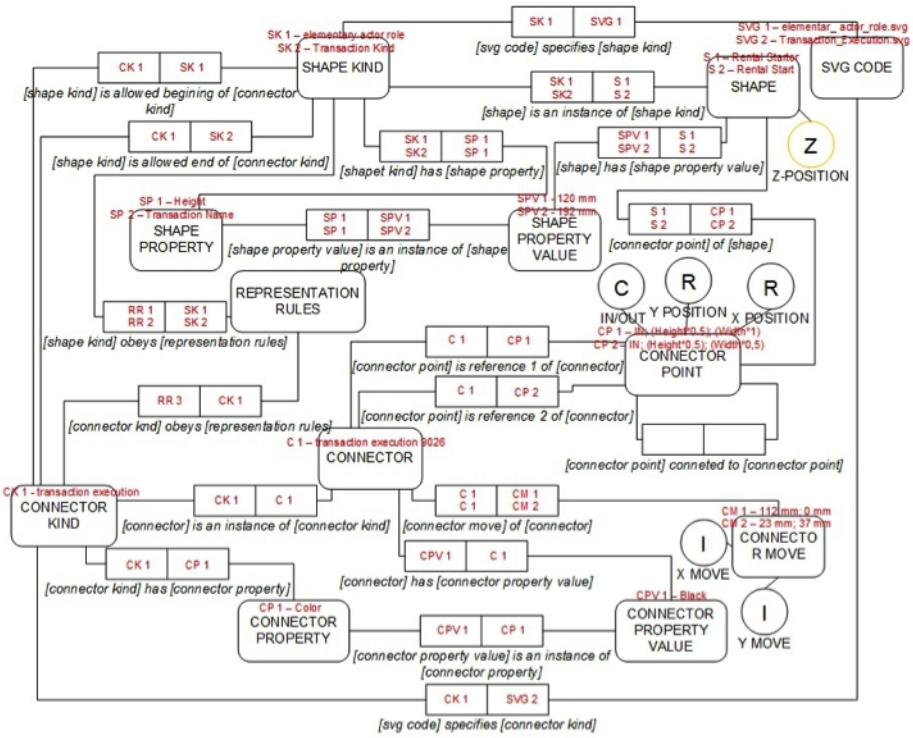


Figure 36: UEAOM - Concrete syntax

Source: Adapted from [27]

CONNECTORS that represent those OAKs and OARKs. For example, a transaction is a black circle with a black diamond inside. It is also according to the REPRESENTATION RULES that a final answer of if an OARK will give origin or not to a connector or if instead it will be represented by the connection of two shape kinds directly. Revisiting the full example from Figure 34, an *elementary actor role shape* would be an instance of class SHAPE KIND, for the representation of instances of the *actor role OAK*.

Transaction shape would also be an instance of SHAPE KIND for the representation of instances of *transaction OAK*. So an instance of class CONNECTOR KIND for the representation of this OAKR would be *transaction initiator connector*, with properties like *line type: dashed*. Many of the SHAPE KINDs and CONNECTOR KINDs are comprised by multiple symbols that need to be considered individually as having a set of properties. Although in most cases the aggregate of composing symbols are treated as "one" in the diagram drafting, such as a circle and diamond in an actor transaction diagram transaction, that have a fixed size (height and width) and none of them can be altered, there are also cases in which symbols need to be treated and moved in the diagrams in a separate and independent way having their own set of SHAPE PROPERTIES or CONNECTOR PROPERTIES like, for example, in a process step diagram where the diamond inside the transaction can be moved and re-sized according to the needs. As a solution for this, classes SYMBOL ELEMENT KIND that specify each symbol element to

be present in a shape kind or connector kind and SYMBOL ELEMENT that are instances of SYMBOL ELEMENT KIND and specify concrete representations of SYMBOL ELEMENTS of a specific kind. As an example of this we can consider the actor transaction diagram SHAPE KIND transaction as being composed by the SYMBOL ELEMENT KINDS *Transaction Diamond* and *Transaction Circle*.

2.6.5 Simplified core classes

Figure 37 presents a simplified version of the UEAOM, with the core classes only. A brief explanation and exemplification of the core classes relevant for the contributions of this project follows. LANGUAGE – used to specify which languages are permitted in the Diagram Editor. Example: «DEMO v3.5». MODEL KIND – each Language can have multiple Model kinds, used to specify which kinds of models are permitted for a certain language in the Diagram Editor. Example: «Construction Model v3.5». DIAGRAM KIND -

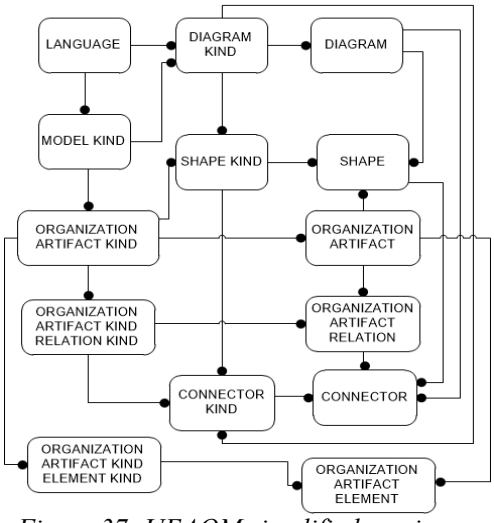


Figure 37: UEAOM simplified version

Source: Adapted from [27]

Each model can have multiple Diagram kinds, used to specify which kind of Diagrams are permitted in the Diagram Editor for a certain Model kind. Example: The Actor Transaction Diagram «ATD v3.5». ORGANIZATION ARTIFACT KIND (OAK) - used to specify which kinds of organization artifacts can be used in models. Example: «ELEMENTARY ACTOR ROLE v3.5». ORGANIZATION ARTIFACT KIND RELATION KIND (OAKRK) – used to specify which kinds of relations are permitted between OAKs. Example: «TRANSACTION KIND.executed by.ELEMENTARY ACTOR ROLE v3.5». ORGANIZATION ARTIFACT (OA) – used to specify a concrete organization artifact instance of a particular OAK. Example: «RENTAL STARTER» is an instance of ELEMENTARY ACTOR ROLE v3.5. ORGANIZATION ARTIFACT RELATION (OAR) – used to specify a

concrete organization artifact relation instance of a certain OAKRK. Example: «CAR PICK UP.executed by.RENTAL STARTER». ORGANIZATION ARTIFACT KIND ELEMENT KIND (OAKEK) – used to specify an element kind of an organization artifact kind. Example: Transaction Name is an element of the Organization Artifact Kind Transaction. ORGANIZATION ARTIFACT ELEMENT (OAE) – used to specify instances of an OAKEK. Example: the string "rental start" that is the name given to a particular transaction that is an instance of the Organization Artifact Kind Transaction.

2.6.6 Conclusion

With this proposal, was possible to confirm that through the multiple application of the type square pattern and the adaptive object model pattern, allows a robust and precise conceptual solution to manage changes in organization artifacts, their models and meta-models, all in run time environment [27].

Even being a novel proposal would be for certain a valid effort to the development of this project.

The idea of using this model for this project starts from the possibility of having the full DEMO aspects models instantiated on it, and could be used for many purposes, as in this case, for the generation of a BPMN workflow model.

CHAPTER 3

BPM Frameworks Comparison

The Business Process Management frameworks are the necessary tools to implement the automated processes and manage the phases of the BPM methodologies and techniques for organizations processes improvement.

A thorough research and analysis regarding BPM frameworks available is made along this chapter. A comparison criteria table is built, a full installation of each of the frameworks (Appendix A-Frameworks installation) is made and a use case example is also implemented (Appendix B-Creating BPMN models) so its installation and implementation details can be replicated and thoroughly understood.

The standard notation used on BPM frameworks is the BPMN[7]. Only BPMN notation frameworks are analyzed and compared.

3.1 BPM Frameworks analysis

In a search for one business process management framework, an exhaustive research throughout the main one's available nowadays is made.

Along this chapter, is possible to learn and be aware of all the pros and cons of each of the founded to be the most appropriated frameworks.

Despite the mentioning of some paid license BPM software editions, only the open-source and free-to-use frameworks are subject of research and analysis according to the defined criteria.

7 BPMN v2.0 is described and explained on Sub-chapter 2.5-BPMN (Business Process Model and Notation)

3.1.1 Comparison criteria

The criteria chosen for comparison follows:

- Import BPMN 2.0 diagrams (.BPMN extensions files);
  - BPMN is the main notation used for the conversion, so it must import the diagrams, independent on the framework used to built them.
- Export BPMN 2.0 diagrams (.BPMN extensions files);
  - As the criteria used for the importation, the framework must allow the exportation of its created diagrams, so they can be imported on other frameworks capable of running BPMN.
- Integrated BPM engine and workflow;
  - The framework must execute the created BPMN diagrams so they can be evaluated in runtime environment.
- Graphically user-friendly interface to create diagrams (Drag and Drop preferably);
  - Bearing in mind that models can be managed, even by technical skill less personal, it must be easy to use and create diagrams almost instinctively.
- Allow the creation of forms for input data;
  - Construction BPMN based DEMO models requires most of the times input of information, so the framework must easily allow the creation of input forms data.
- Allow the use of external database for querying and store external data related to the purpose of the process;
  - It must be possible to create connections to external databases so actual informations systems already using databases can be used to perform and execute the BPMN diagrams created.
- Possibility of running and debugging the diagrams created on the go;
  - Having the created processes, it must allow to execute them so a kind of an evaluation step-by-step with debugging can be made to the created process.

3.1.2 ProcessMaker

About

The *ProcessMaker* (PM) is an open source Business Process Management (BPM) and a workflow software designer. Is used by small medium-size organizations and business. *Colosa Incorporation* performs its development and maintenance [30].

It allows organizations to optimize the business complex processes operations and workflow management, and put them ready for a certain diversity of people to take part on important decisions of the processes [30].

Key Features

Bellow are shown the major features for a user-friendly point of view usage of the PM software: [30]

- Drag-and-drop, browser-based interface, which requires no prior programming experience;
- Management of users, groups, forms, documents, messages and alerts with the click of a button;
- Workflow maps in a graphical interface with drag-and-drop objects;
- Custom forms in the Dynaform Editor based on XML, with optional editing in HTML and JavaScript;
- Custom output documents in PDF or DOC formats, created in a WYSIWYG (What You See Is What You Get) page editor;
- Triggers with optional PHP code to perform complex calculations and advanced functionality;
- Functions in Windows, GNU/Linux or UNIX, on top of a standard stack of Apache, MySQL and PHP (WAMP/LAMP);
- Coded in PHP, using Smarty templates, Propel database connectivity, and ProcessMaker's custom Gulliver development framework;
- SOA compliant web application, offering web services based on the Web Service Definition Language 1.1 protocol.

Software Editions

There are three main editions of the software; ProcessMaker Open source Community Edition; Cloud Edition; On-Premise Enterprise Edition.

The On-Premise Enterprise Edition could be tried for a limited period of time (30-days), after that, a paid license must be acquired in order to use it normally and in its full potentiality [30].

The ProcessMaker Open Source Edition is recommended for developers:

- Includes Core BPM and Workflow features;
- No patches upgrades available for migration to new releases;
- No software fixes or patches provided;
- No workflow templates;
- No enterprise features or plug-ins;
- No support;

- GPL v3 license.

ProcessMaker Enterprise "On-Premise" subscriptions are recommended for corporations, governments and other organizations looking for an enterprise scale, production-ready business process management software that can be rapidly deployed on-premise:

- Full Featured BPM & Workflow Automation;
- Available in a variety of Enterprise plans;
- Enterprise features and Plug-ins;
- Patch upgrades provided for easy migration to new releases;
- Web, Email, and Phone Support;
- Optional 24x7x365 Performance Monitoring;
- Workflow Solution Templates;
- Indemnity and Warranty;
- Commercial license.

The ProcessMaker Cloud Edition is recommended for corporations, governments and other organizations looking for an enterprise scale, production-ready business process management software that is hosted in Tier One Cloud facilities (Amazon EC2, EAPPS, etc):

- Full Featured BPM & Workflow Automation;
- Enterprise features and Plug-ins;
- Workflow Solution Templates for increased Productivity;
- Unlimited Trouble Tickets and unlimited phone support;
- 24x7x365 Performance Monitoring;
- Indemnity and Warranty;
- Governed by their terms of service.

Summary

According to the comparison criteria the PM framework fails on a few of them. Since and for the sake of this project, only the free versions can be used, the free edition of PM is extremely limited.

The manual installation on a Unix/Linux machine is complex and the lack of information about it, for instance in a community forum, does not help it.

The BPMN models import and export on the free version fails on one of the most important criteria, and only provides the export and import of PM extension type files.

After a meticulous and final analysis, the PM framework is not one of the

chosen ones for the second part of the project.

3.1.3 jBPM vs Activiti

About

Since jBPM and Activiti are both BPM frameworks that have some similarities and were developed by the same developer-teams, the decision of creating a comparison between them, regarding its differences and its equalities seemed appropriated. Actually, the main developers of Activiti today, were the ones related to jBPM development in the past, meaning these two frameworks are development related somehow [31].

Both of these tools are developer-oriented, built around the concept of a state machine and also implementing the BPMN 2.0 specification. With this, the possibility of finding similar functionality between them is high. Besides its similarity, there are some differences, and these, are important to mention and can be seen on table 2.

Activiti started on 2010 by Tom Bayens and Joram Barrez. The latter was the former founder and core developer of jBPM (JBoss BPM). Acts as an independent open source project but is related and funded by Alfresco [31].

Activiti is able to deploy process business definitions, start new projects instances, execute user tasks, and perform other BPMN 2.0 functions.

After version 5.11 (included), Activiti provides a way of creating new BPMN models without the need of the Eclipse IDE plug-in using the incorporated tool, Activiti Modeler by KIS BPM.

jBPM stands for Java Business Process Management. Aside with the BPMN 2.0 (Business Process Model and Notation version) is has its own process (jPDL) definition language in the earlier versions.

Its maintenance and actual development is responsibility of the JBoss Community and is released under the ASL.

Main differences between Activiti and jBPM:

Table 2: Main differences between Activiti and jBPM

Source: Adapted from [31]

Description	Activiti	jBPM
Community Members	Mainly alfresco employees compose the Activiti development team. There are also some individual developers.	Composed by a team from Jboss employees. Also there are individuals developers.
Business Rules Support	Provides a basic integration with the Drools	jBPM and Drools are integrated on a project

	rule engine to support the BPMN 2.0 business rule task.	level.
Additional tools	Process definitions can be defined (drawn) using the Eclipse designer IDE plugin. Before version 5.11 provides the Activiti Modeler by BPM KIS allowing the possibility of creating and editing BMPN models. Also provides the Activiti Explorer, which is a web interface used to start new processes, work with tasks and forms, and manages running processes.	Provides a modeler based on the Oryx Project and also the Eclipse designer plug-in. Also has a web application, which can be used to start new processes instances and work with the tasks.
Project	Has got a very strong development community. Its main components are Designer, Explorer and REST application	Also has a very strong developer and user community. However the release schedule for the main components (Eclipse IDE) is not crystal clear as is on the Activiti.

Software Editions

Full version of both softwares are open source and free to use.

Today's stable and current version of jBPM is 5.4 and can be downloaded from the official web site from JBoss Community.

Activiti current version is 5.11 and includes the self Activiti Modeler. This allows the possibility of simple editing the BPMN diagrams directly on the Activiti Explorer without the need of using Eclipse Designer Plug in. However, the use of Eclipse Designer plug-in is advisable, since it provides a better way for complex edition and also the possibility to create java classes to test the correctness of the processes before deployment and consequent importing to the Activiti Explorer.

Summary

It is always a difficult task to choose in-between very similar frameworks, the most suitable for the job. In this case, both were created almost by the same team for the same purpose. That's why at this point, was important to stay only with one of them in order to continue the analysis.

According to Activiti user-guide, the Activiti started with the version 5, for a specific reason. Since the Activiti development team is mainly composed with the ones who in the past started the jBoss framework (version 1 to 4), it is somehow explicit, that Activiti framework is a continuous development work, from the previous work done in the jBoss.

With this, is appropriated to choose the Activiti framework, for a more detailed analysis according to the criteria.

Besides being a developer-oriented framework, it is a very complete one. Allows the developer to be in total control of the situation. For the end-user this could be a problem however and having in mind its powerful capabilities, is appropriated to give it a try on a real case example situation.

3.1.4 Bonita Open Solution

About

Bonita Open Solution was created in 2001, since then and until 2009 was kept under its original creators, the French National Institute for Research in Computer Science [32]. Only in the year 2009, one company dedicated to its type of activity, supported the development of Bonita Open Solution, the BonitaSoft Corporation. It is an open source business process management and a workflow engine based on java and relies on Eclipse to work properly. The Bonita Open Solution consists in three main parts: Bonita Studio; Bonita BPM Engine and Bonita User Experience. The Bonita Studio allows the user to graphically create and modify processes accordingly the BPMN standard. Connections to different type of database's can also be made without much effort since it is all based on drag-and-drop interaction, along with the creation of forms and documents inputs. The Bonita BPM Engine is a Java API that gives the user the possibility to interact programmatically with the created processes. The Bonita User Experience is a web service portal that permits the end user to manage and take part on the processes [32].

key Features

- Graphical business process model notation design (BPMN 2.0);
- Drag-and-drop user-friendly interface, which requires no prior programming experience;
- Import of BPMN 2.0 diagrams (e.g. Eclipse Design Plug in);
- Connections to database (MySQL, oracle);
- Forms, input documents, input data;
- BPMN running and debugging on the go.

Software Editions

The Bonita Open Solution is open source and can be downloaded under the GPL license for the free Edition. Enterprise Editions are also available. The next table shows the Editions available.

Table 3: Bonita Open Solution Editions Available

Source:Adapted from [32]

	Open Source	Teamwork	Efficiency	Performance
Core BPM Suite features: process modeling, connectors, simulation, web forms, application generation, User Experience portal, and more	X	X	X	X
Collaboration with shared developer repository		X	X	X
Advanced productivity for developers - LDAP synchronization - Re-usable forms and form field widgets - Customizable look and feel - Connectors inside web forms for dynamic interaction - Graphically-based wizards for web services, Salesforce, SQL, and SAP connectors		X	X	X
Advanced productivity for business users - Process optimization - Generate process documentation - Install custom Kis and create dashboards		X	X	X
Document Management		X	X	X
Process Templates: HR, Finance, Quality, IT and more		X	X	X
Monitoring			X	X
Error management				X

Summary

BOS (Bonita Open Solution) is from the above frameworks analyzed, the one, which totally stands out. Its not developer-oriented, actually it is extremely

end-user oriented. One does not need to have programming or even high computer capabilities skills to use it. The creation of BPMN diagrams is simple, along with running and debugging the processes diagrams.

The drag-and-drop interface facilitates the end-user interaction with the software and in a manner of minutes is possible to create and execute a process diagram with data input (forms) and database connections (querying and storing data related to the process) without much of an effort.

With this, the BOS is one of the chosen ones to be tested with a simple case example.

3.2 Use Case BPMN Implementation

After the first analysis of the frameworks, it is appropriated to have them tested with a simple use-case example. A BPMN model, based on a case example scenario is created and executed on each of the frameworks.

3.2.1 Case Description

An implementation of a simple BPMN model based on EU-Rent[8] were created and executed on each of the chosen frameworks for this second part of the analysis.

This case is based on the EU-Rent, but without some particular details. It will only regard the process of booking a rent in advance. Could be as a "walk in" customer or over an online/local platform, storing the selected options for the rent (start/end date, car type, pick-up/drop-off branch), checking the availability of the car for the desired dates and finally storing into the database all the related information about the renting.

The importance here is to have a BPMN model ready to be executed on each of the frameworks in a way to analyze its functionality and the ease of use by creating from scratch an executable BPMN model. With this is possible to analyze the frameworks according to the criteria.

3.2.2 BPM framework implementation

The BPMN as a transformational process modeling language is method-independent, so that, different BPMN models and processes can be created having at the same time exactly the same meaning and functionality. That's why a different BPMN model based on the same scenario (EU-Rent) for each framework is made, since both frameworks have its on way of creating BPMN models.

8 The EU-Rent case description is made on Sub-chapter 5.1 - Eu-Rent case description

3.2.2.1 BOS Implementation

As mentioned on the BOS description, it has a drag-and-drop user interface for creating or editing BPMN models. The creation of new models from scratch is straightforward even with database connections and data input forms.

Data input forms were used as well as a database connection. In this particular case is used a MySQL database and connection. The MySQL database was previously created in order to store the related renting information and the company related data, as the available cars and branches.

With the details mentioned above in mind the next step was about creating the model.

BPMN model

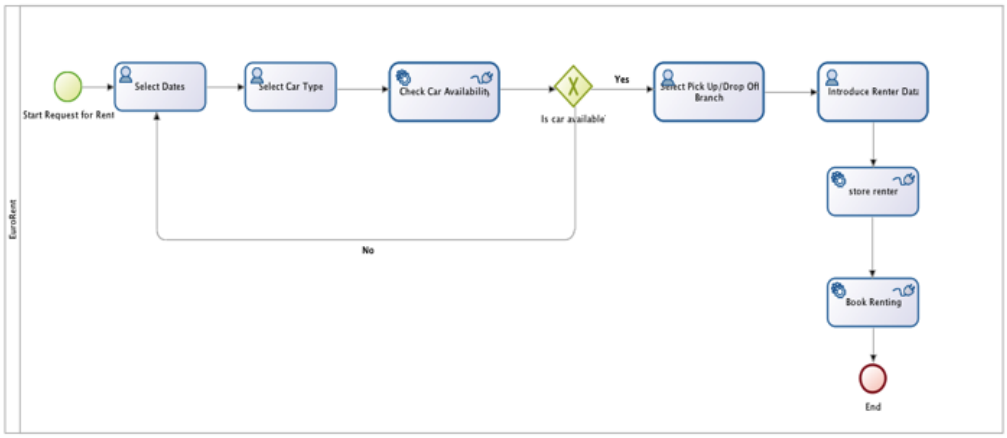


Figure 38: BPM Implementation on BOS - BPMN model

The creational details of the BPMN's processes on BOS can be seen on the Appendix B.1-Creating BPMN models on BOS as well as the finished running model.

Based on the Case Description, the created model is appropriated to the stated problem.

BPMN Model description:

The request for booking a rent starts with the "walk in" customer or the employee requesting it on the online/local platform. On the first process "Select Dates" the user should chose the start/end date for the renting. On the second process "Select Car Type" the user should chose the desired car type.

The third process is a system process, and checks on the MySQL database if the desired car for the selected dates is available.

If not available, returns to a new introduction of start and end dates and car type. If available, continues for the next process.

The fourth process the user selects the Pick-up and Drop-off branches.

The fifth process the user introduces the person personnel information (name, address).

The sixth process connects to the database and saves the renter information.

The seventh and last process stores the renting options.

BOS implementation Pros and Cons

At first, it was difficult to deal with the application form and global variables. Since each process has the possibility of having its own variables, those variables stay on the processes becoming unreachable outside of it, so all those variables must be created as global variables so they can be accessed outside the process and be used on other processes.

The synchronization of the application forms with each process was also and at first, an issue because the main pool of the main model needed to be skipped or it was not possible to start and run the model.

Figuring out these simple issues the major implementation was really straightforward. The drag-and-drop user interface is helpful and finding the menus and the desired options is simple as well.

3.2.2.2 Activiti Implementation

There are two ways of creating a new BMPN model on Activiti. First way, is using the IDE Eclipse Designer plug-in, the other is using the incorporated Activiti Modeler on the Activiti Explorer (Available only after version 5.11 included). The latter has an easy setup, since it is incorporated on Activiti Explorer, and is very easy to reach it, however its usage and at first sight, look difficult to understand.

For this purpose is going to be used the Eclipse Designer plug-in to construct the BPMN model.

Having in mind all the mentioned details about the scenario, various attempts on Eclipse Designer were made in order to reach an acceptable model that could be executed and verified on Activiti Explorer.

According to Activiti User Guide all seems extremely simple, however and for an informatics skill less user, things could get very hard to understand.

In this chapter is not going to be shown all the models created. Only the most acceptable compliant model is going to be taken into account for the next steps.

BPMN model

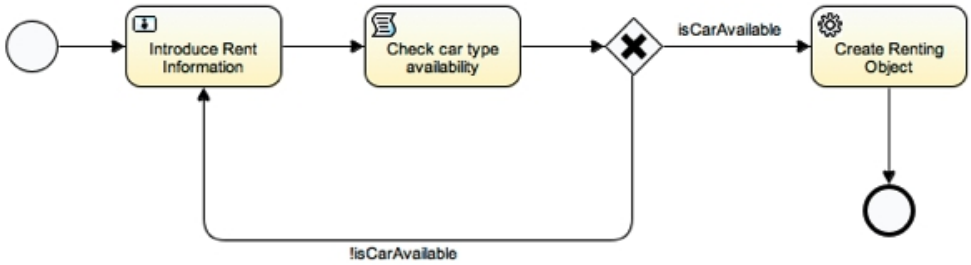


Figure 39: BPM Implementation on Activiti - BPMN model

BPMN model description

All the renting information is introduced in only one process. The "Introduce Rent Information". Based on the information introduced, the "Check car type availability" checks if the desired car is available for the introduced dates. If available, proceeds to the next process, if not available, goes back to the previous process "Introduce Rent Information". The "Create Renting Object" process will create a Java Object with all the renting data introduced and store it on the Activiti database.

Activiti implementation Pros and Cons

Opening the IDE Eclipse Designer for the first time, the first impression is something like "How and from where should I start". There isn't any guide or How-to to help the user to start. With this, all the things seem extremely hard to find and to understand. There is not a way to easily create database connections or even input data forms.

If the user does not have any Java Oriented Programming skills, it is a problem, since debugging models are made with java classes and objects.

One of the good things about IDE Eclipse designer is the possibility to execute the already created model in eclipse so then the user can have sure the model will work on Activiti Explorer. But even this possibility is not an easy thing to do if the user does not have any java programing skills.

Bottom line, Activiti Designer is totally developer oriented not suitable for inexperienced users.

3.3 Summary

3.3.1 Frameworks comparison according criteria

According to the initial criteria, a comparison table was created.

Table 4: Frameworks comparison criteria

	ProcessMaker	**Activiti**	**BOS**
Import BPMN 2.0 diagrams	No	Yes	Yes
Export BPMN 2.0 diagrams	No	Yes	Yes
Integrated BPM engine and workflow	Yes	Yes	Yes
End-user friendly interface	Yes	Yes	Yes
Form creation for input data	Yes	Yes	Yes
Process diagram creation	Yes	Eclipse IDE plug-in	Yes
Connection to external database	No	No	Yes
Running and debugging processes	No	Yes	Yes

3.3.2 Analysis conclusion

Having in mind the initially criteria specially created for the purpose of this analysis, things get a bit easier on selecting one of them.

All frameworks tested have in common the possibility of creating BPMN models; ones more Developer-oriented (Activiti, jBOSS) others more user-friendly oriented (BOS).

According to the final table comparison criteria is clear that the BOS is the most appropriated framework for the sake of this project.

In summary, BOS framework is the most suitable for the purpose of this thesis according to the initial criteria.

CHAPTER 4

From DEMO to Workflow

The major goal of this graduation project is to convert an UEAOM based DEMO organization business models into a compliant UEAOM based BPMN process without losing any kind of relevant information and meaning on the process. The main idea is to gather the essential within the available information from the DEMO models and generate a compliant BPMN process.

For that ending and as previously started, a thorough analysis to the available BPM frameworks was made, along with a comparison between them. With that an insight of the DEMO methodology, constructions and its foundations. The UEAOM proposal and the BPMN notation was also point of insight.

With all these concepts acknowledged and being able to gather all that knowledge, would be for certain possible to workout a valid way and method to accomplish the desired conversion.

In this Chapter the essential construction requirements for both DEMO and BPMN were analyzed. How they are constructed? What kind of information is needed for both constructions? What is necessary for a valid construction of both models? Questions that are answered along this Chapter.

Regarding DEMO models, they were analyzed in a search for a concise and a kind of creation pattern, a pattern that could work as a rule for gathering the important and the relevant information available within the DEMO models.

4.1 Construction requirements

Both DEMO and BPMN models are totally different in its notation and

construction method. Because of that, both construction methods needed to be thoroughly acknowledged and understood.

4.1.1 DEMO models

DEMO models are made to be easily understood and read by humans. With this methodology humans are capable of visualize an organization by graphical diagrams, such as Organization Construction Diagram (OCD), Process Structure Diagram (PSD) as seen on DEMO methodology[9]. These models are platform-independent and are easily designed using a diagraming tool such as Microsoft Visio®. Along with the platform-independent, there is not an available way to save and propagate possible changes along the entire model in runtime environment.

Making DEMO models interchangeably into any other format is not even close of being straightforward. The ideal situation would be the possibility of gathering all the models into one big model so that all the information regarding such enterprise and their processes should be in the same place ready to be gathered and workout for different types of outputs. So far and with the DEMO models, that situation is not possible, even with the three cross-model tables that collect information from different models, however the captured information is not enough and not suitable for direct use on the conversion.

4.1.2 BPMN models

Using BPMN as a standard for capturing and representing business processes is definitely valuable and helpful for any organization however and because of being extremely wide open regarding its notation components and due to its lack of defined semantics on its method construction the process of choosing the right components for the conversion rules was not particularly easy. The wide range of options for its construction and even the possibility of mixing constructions allows the creation of models with semantic errors [24]. These type of errors in the early phases of systems development are normally the most serious and the most difficult to find and solve so they must be avoided. There is not a pattern or solid semantics for the construction of BPMN diagrams, meaning that for one particular problem could be constructed various models, however valid. For instance and regarding processes where a simple task followed by other tasks can be used, ones can simple make use of an Event Subprocess with all the tasks inside of it. This type of approach can be seen as more clean since all the tasks are gathered inside an Event that can be minimized or expanded to the user preference however and for others it can be seen as an increase of complexity since it could be interpreted as an independent process when expanded. This situation led to ambiguity on BPMN models construction.

9 DEMO methodology is described on Sub-chapter 2.2-DEMO Methodology

For those reasons an ability to have a statically pattern for BPMN construction would be extremely important and a valid help for this type of conversion.

4.2 Acknowledging DEMO models

Regarding DEMO, from the ontological aspect models, special attention to the Action Model was gave, which is the one that reveals and specifies major details not found in all other models. The Action Model consists in the specification of action rules, that show how actors deal with their agenda on realizing their actions of certain transactions. In other words, they specify all transaction acts, the actor roles responsible for those acts, the trigger of each transaction, the conditions of the state of the world that have to be checked and that influence flow and new facts that are created or facts that have to be fetched from some fact bank or from one of the actors. Other models that seemed interesting for this matter of conversion from DEMO to BPM were the Construction Model, namely because of the information about the links between actors and the transaction that they initiate and/or execute; and also the links with the information banks. Nevertheless, was concluded that the Action Rules would be the main source of information in the conversion process, as they specify all agendum for each of the internal actors of the organization and for all the transactions. And all agendum is the starting point to find BPMN process requirements and structure the modeled processes themselves.

DEMO models construction patterns

Based on what was analyzed concerning the AM (Action Model) and the DPM (Demo Process Model), some interesting patterns that can be used for the BPMN construction were found.

Regarding the TRT (Transaction Result Table) in the DPM , each transaction have a kind of standard construction composed by various steps, as seen on the definition of the ontological model[10]. Some of the steps that constitute an entire transaction pattern could happen to be as a hidden step of the transaction definition, but somehow it exists implicitly.

Table 5 depicts exactly how its construction work:

Table 5: Transaction standard construction pattern

Step	Actor role
C-act request	requester
C-fact request	
C-act promise	addresser
C-fact promise	

10 This definition is shown on Sub-chapter 2.1.2-The PSI-Theory.

Production-act	addresser /producer
Production-fact	
C-act state	addresser
C-fact state	
C-act accepted	requester
C-fact accepted	

This pattern shows all the operation axioms (C-acts/P-acts and C-Facts/P-Facts) that a transaction have, even that some of them could be indirectly omitted as acts being done tacitly.

With the same idea in mind for the seek of construction patterns, a kind of syntax has been found for the action rules. Table 6 shows its normal construction pattern:

Table 6: Action rule standard construction pattern

WHEN	One or More operation axiom of transaction T (C-fact or P-fact)
Possible THEN	One or More operation axiom of transaction T (C-act or P-act)
Possible WITH	One or more informal/formal conditions
Possible IF, FOR, WHILE condition	One or more formal/informal conjunction(AND)/disjunction(OR) conditions
THEN	One or More operation axiom of transaction T (C-act or P-act)
Possible WITH	One or more informal/formal conditions
ELSE	One or More operation axiom of transaction T (C-fact or P-fact)

It is known that different organizations have different type of actions rules but its construction pattern occurs exactly the same way, as it speaks for itself, action rule as a rule for the actors taking action.

The finding of this pattern is important for the development that is being taken here since for the BPMN construction a certain kind of construction pattern has to be found. If all the action rules have the same construction pattern and based syntax, the BPMN would be always constructed based on that pattern, ensuring that information was not loss on the process.

Was concluded that the Action Rules, part of the Demo Action Model would be the main source of information in the conversion process, as they specify the *agendum*[11] for each of the internal actors of the organization and for all

11 By Agendum means things/acts to be done like several items part of an agenda.

transactions.

4.3 DEMO essential conversion information

Transactions

Transactions are the first available type of information, that results from the appliance of the DEMO methodology, where is possible to find essential information about what the organization do and how they do it. Each transaction can be gathered together and the TRT (Transaction Result Table) is appropriated since it shows for each transaction its kind and product/result kind.

Actor and Actor Role

With transactions in hand came the ones responsible for taking part of those transactions. People makes any organization in good, or not, working order. With workflow processes happens the same.

The actors are responsible for making processes executable just by performing their ordinary actions. Having the actors and its related transactions is fundamental for this type of conversion. Also in the TRT, the Initiator and the Executor actor of each transaction are depicted.

Action Rules

The action rules are the main repository of information that is used as the base for the DEMO conversion into BPMN. However and looking for the information that could or not be gathered from the action rules, it is impossible to understand, and depending of how they are created, since they have optional fields, that they may vary a lot, from organization to organization. This means that the action rules must be restricted to a kind of creation pattern so ambiguity on its creation is avoided.

4.4 Conversion rules between models

It is clear that there must be conversion rules so that DEMO models can be converted into BPMN models in the most automated way possible maintaining total equivalence between them.

After several experiments of converting DEMO models to BPMN models was found that some several ambiguous elements from BPMN had to be discard – e.g., the message element that easily becomes redundant with tasks – and arrived at the following conversion rules – one of the main contributions of this thesis – so that each DEMO concept has a 1 to 1 correspondence to a BPMN concept. The format used was: **DEMO concept < > BPMN concept**.

Because DEMO has a strong semantics with a comprehensive meta-model, these proposed rules imply that by using the few BPMN elements that were selected a more precise semantics for BPMN was possible when compared to an unrestricted use of it or to using BPMN as a starting point to model enterprise processes.

Transaction < > Pool

Each transaction is represented on BPMN as a Pool, and inside of each pool only its related coordination and production acts/facts. Each transaction must start with a *start event* and be finished with an *end event*. A *start event* is triggered by other coordination or production act of any transaction. The transaction ending, can happen by several ways. For instance: by a *revoke request act* enacted by a certain actor.

Actors < > Lane

Actors are associated with transactions, so each transaction have its correspondent actor. All the events depicted inside a lane are responsibility of its respective actor. Actors initiate and/or execute transactions, so each transaction is represented by two lanes and all the events depicted inside a lane are responsibility of the respective actor.

Actions, Flows, Conditions and Condition Evaluation < > Tasks

Actions, Flows, Conditions and the respective conditions evaluations (in action rules) are represented on BPMN as tasks. Tasks have an input and an output flow. Each actor is responsible for the tasks found in their respective lane. Depending on the implementation details, a task can be manually performed by its actor or simply performed by the system, however the implementation details on how tasks are performed are not the main focus at this phase of the project.

Coordination-facts/Production-facts < > Signals (Throw and Catch Event)

Coordination facts or production facts are converted to signals in BPMN, meaning that they are used as *throw* or as a *catch* signals. This is a key conversion rule as it allows to “isolate” the specification of each transaction and their respective actor's rules in a pool and the occurrence of particular facts will possibly enact one or more transactions at the same time. This gives a high degree of flexibility and modularization.

Sequence flow < > Gateways

The sequence flow between the BPMN construction could be thought as simple as connecting tasks and events and so on, however some careful must be taken since there are many ways on BPMN notation of splitting and merging sequence flows from tasks and events. In this case there are going to be used two types of gateways: the parallel (+) and the exclusive (X) gateway.

Depending on how actions, conditions, its respective evaluations (tasks) and events (Start, End, Throw and Catch events) are sequentially made, gateways must be used according the action rule construction.

4.5 Conclusion

At this level of the project it is fair to announce that some good discoveries were made related to the requirements of each type of model. Namely what is relevant to gather from the DEMO aspect models and - the most innovative contribution – the model conversion rules.

Besides this, the best way to confirm if it all makes sense or even if it works, is definitely with the evaluation of the previous guidelines using a use case as example to test it.

For now, this must be seen as guidelines to be followed on the following steps of the entire generation process.

CHAPTER 5

Workflow Process Generation

The best way of evaluating the previously suggested guidelines and conversion rules is implementing them directly on a use case. This chapter aims to evaluate the generation of a BPMN process based on the DEMO models of an organization based on [33], where the DEMO methodology[12] was thoroughly applied.

In the previous chapter was found what kind of information is needed, which is the more appropriated and how the models should be crossed, using the conversion rules that led to the usage of clear semantics to its construction.

As previously described on Sub-chapter 2.6, the UEAOM is a conceptual schema being used as the base for the effort to create a wiki-based system that allow an effective enterprise modeling. So based on the UEAOM and its capabilities of modeling a valid instantiation of the action rules are depicted.

This Chapter also shows, at the same time that guidelines were being tested, their evolution according the necessity of the real case using for that the research method - Design Science Research presented by A. R. Hevner - depicted on Sub-chapter 1.4 until a valid output was reached.

5.1 Eu-Rent case description

Based on [33], the EU-RENT case is described as follows:

"EU-Rent is a company that rents cars to persons, operating from

12 DEMO methodology is described on Sub-chapter 2.2-DEMO Methodology and more detail can be seen on [11]

geographically dispersed branches. The cars of EU-Rent are divided in car types (brands and models); for every car type there is a particular rental tariff per day.

A car may be rented by a reservation in advance or by a 'walk-in' customer on the day of renting. A rental contract specifies the start and end dates of the rental, the car type one wishes, the branch where the rental starts (called the pick-up branch), and the branch where the rental will end (called the drop-off branch). Rentals have a maximum duration.

The person who rents the car is called the renter. The one who is going to drive is called the driver. A rental will only be started if the driver has a valid driving license. In addition, a car of the requested type must be available.

As soon as the car of a rental has been dropped-off, the rental can be ended, after the incurred charge has been paid. This charge may consist of several elements. First, there is the basic charge (number of days times the tariff per day). Next, there may be a penalty charge for exceeding this duration (number of extra days times the late return penalty tariff). Lastly, a location penalty charge is added if the car has been dropped-off at another branch than agreed (this charge depends on the distance between the branches)."

Based on this case description, DEMO methodology was thoroughly applied and all the models have been constructed. Before its final models, some others versions were created as a draft to have at the end the final ones. All models can be consulted for further details on [33].

5.2 Applying guidelines

Transaction result table and actors

The transaction result table enumerates all the transactions and its results kind. Each color represents one transaction. They were found by the application of the DEMO methodology as a first approach to the description of the use case. Along with the transactions the responsible actors and their roles on the transactions are also described as the initiator and the executor.

Table 7: Eu-Rent - Transaction result table

		Transaction kind	Transaction result	Initiator	Executor
	T01	rental start	[rental] has been started	renter	rental starter
	T02	rental end	[rental] has been ended	renter	rental ender
	T03	car pick-up	the car of [rental] has been picked-up	rental starter	driver
	T04	car drop-off	the car of [rental] has been dropped-off	rental starter	driver
	T05	rental payment	[rental] has been paid	rental ender	payer

Action rules

As seen on Sub-chapter 2.2 is in the Action Model that takes place the creation of the action rules. There is a standard syntax and method[13] to construct them, and making use of that syntax, the following rules depicted on Figures 40, 41, 42, 43, 44, 45, 46, 47, 48, 50, 49 were constructed for the EU-Rent use case example:

B-A01 rental starter 1	WHEN	rental start of [rental] is requested		
	if			driver has valid driver license
			AND	there is a car available of the car type of rental
			AND	(contracted end date minus contracted start date) <= max rental duration
	then		rental start of [rental] must be promised	
	else		rental start of [rental] must be declined	

Figure 40: EU-Rent - Action rule 1

B-A01 rental starter 2	WHEN	rental start of [rental] is promised	
	with	car pick-up of [rental] must be requested	
		with	requested settlement time is within contracted start date
		and	the car of [rental] is [selected car of car type of rental]
	and	car drop-off of [rental] must be requested	
		with	requested settlement time is within contracted end date

Figure 41: EU-Rent - Action rule 2

B-A01 rental starter 3	WHEN	car pick-up of [rental] is promised
	and	car drop-of of [rental] is promised
	then	rental start of [rental] must be executed
		rental start of [rental] must be stated

Figure 42: EU-Rent - Action rule 3

B-A01 rental starter 4	WHEN	car pick-up of [rental] is stated
	then	car pick-up of [rental] must be accepted

Figure 43: EU-Rent - Action rule 4

B-A01 rental-starter 5	WHEN	car drop-of of [rental] is stated
	with	the actual drop-off branch of [rental] is [branch]
	then	car drop-of of [rental] must be accepted

Figure 44: EU-Rent - Action rule 5

B-A01 rental starter 6	WHEN	rental start of [rental] is stated
	then	rental start of [rental] must be accepted

Figure 45: EU-Rent - Action rule 6

13 The Action rule standard syntax is described on Sub-chapter 4.2-Acknowledging DEMO models

B-A02 rental ender 7	WHEN	rental end of [rental] is requested			
		if		[rental] has been started	
			AND	the car of [rental] has been dropped-of	
				OR	the car of [rental] has not been picked-up
		then	rental end of [rental] must be promised		
		else	rental start of [rental] must be declined		

Figure 46: EU-Rent - Action rule 7

B-A02 rental ender 8	WHEN	rental end of [rental] is promised	
	then	rental payment of [rental] must be requested	
	with		the final charge of [rental] = [money]

Figure 47: EU-Rent - Action rule 8

B-A02 rental ender 9	WHEN	rental payment of [rental] is stated		
		if		everything is ok
		then	rental payment of [rental] must be accepted	
		else	rental payment of [rental] must be rejected	

Figure 48: EU-Rent - Action rule 9

B-A02 rental ender 10	WHEN	rental payment of [rental] is accepted
	then	rental end of [rental] must be executed
		rental end of [rental] must be stated

Figure 49: EU-Rent - Action rule 10

B-A02 rental ender 11	WHEN	rental end of [rental] is stated
	then	rental end of [rental] must be accepted

Figure 50: EU-Rent - Action rule 11

Based on the previous guidelines was concluded that the Action Model is the perfect gathering of the information needed for the conversion. However, the action rules depicted above, using the standard syntax does not have the enough rigidity to work as a rule for the conversion, bearing also in mind the BPMN construction requirements. So that, a new DEMO action rule syntax is proposed and depicted on the following chapter.

5.3 New DEMO Action Rule Syntax proposal

As previously seen, it was clear that the actual syntax is not complete enough to provide a valid BPMN construction.

Figure 44 shows the action rule that is used as the base to workout the new DEMO action rule syntax proposal since it is the most complex, compared to the others available, found on this particular case. It is a perfect example on how Action Rules are “the neglected son” of DEMO Action Model. In real life many different conditions and facts have to be verified before one can proceed to accept the drop-off of a car. In this action rule specification, the only verified fact is if the branch where the car is delivered is the same as the contracted one but no action is specified for the contrary case. Also in this rule

was found a common problem in DEMO's Action Rule Meta-Model, namely, what is the meaning of the construct *with*? *With* is found in many action rules and apparently with different functions: creating new facts, verifying new facts, etc. Indeed, in the most current public version of DEMO, it is assumed that "the syntax and the formal semantics of the action rules need to be elaborated yet" [34]. It is valuable to have models that abstract from infological and datalogical aspects as well as implementation issues, like DEMO's Construction Model, Process Model and State Model do. However, DEMO models can never be fully independent of implementation and resource constraints from the organization's reality. Rather, at most, they are implementation abstracted [35]. Action Rules are definitely the perfect spot to make the bridge between the most implementation abstracted views of an organization – like the transactions and actor roles of the Actor Transaction Diagram – with the implementation world. This is because, while thinking on the flow and requirements for the action of actor roles, it is inevitably needed to think about necessary fact evaluations, information requests, data storing etc. So why not specify such items while devising DEMO action rules? In this manner, the ontological model of an organization fully guides the specification of relevant infological, datalogical and implementation requirements. Following the design science research method mentioned in Chapter 1.4 - Research method – is presented in Figure 51 the action rule that is the final result of the evolution of the simple action rule depicted on Figure 44. All re-written final rules using the new syntax can be seen on Appendix C.1-EU-Rent action rules new syntax.

AR05							
WHEN	car drop-off of [rental] is stated	C-FACT	ARCW01 (ARCET01)				
ACTION (FIRST, ATOMIC)	Read contracted car drop-off details	READ_DATA	ARCA01				
ACTION (FLOW)	**if** the actual drop-off branch of [rental] is [contracted_branch]	fact evaluation	ARCA02 (ARCIF01(ARCC01))				
	then ACTION (FLOW)	**if** the actual date is [contracted drop-off date]	fact evaluation	ARCT01 (ARCA03(ARCIF02(ARCC02)))			
		then ACTION (FIRST, BLOCK)	ARCT02(ARCA04)				
		ACTION (FIRST, ATOMIC)	penalty charge == false	WRITE_DATA	ARCA05		
		ACTION (AFTER, ATOMIC)	car drop-off of [rental] must be accepted	C-ACT	ARCA06		
		else ACTION (FIRST, BLOCK)	ARCE01(ARCA07)				
		ACTION (FIRST, ATOMIC)	penalty charge == true	WRITE_DATA	ARCA08		
		ACTION (AFTER, ATOMIC)	late return penalty charge == true	WRITE_DATA	ARCA09		
		ACTION (AFTER, ATOMIC)	car drop-off of [rental] must be accepted	C-ACT	ARCA10		
	else ACTION (FLOW)	**if** client accept to delivery it anyway	condition evaluation	ARCE02 (ARCA11(ARCIF03(ARCC03)))			
		then ACTION (FLOW)	**if** the actual date is [contracted drop-off date]	fact evaluation	ARCT03(ARCA12(ARCIF04(ARCC02)))		
			then ACTION (FIRST, BLOCK)	ARCT04 (ARCA13)			
			ACTION (FIRST, ATOMIC)	penalty charge == true	WRITE_DATA	ARCA14	
			ACTION (AFTER, ATOMIC)	location penalty charge == true	WRITE_DATA	ARCA15	
			ACTION (AFTER, ATOMIC)	car drop-off of [rental] must be accepted	C-ACT	ARCA16	
			else ACTION (FIRST, BLOCK)	ARCE03(ARCA17)			
			ACTION (FIRST, ATOMIC)	penalty charge == true	WRITE_DATA	ARCA18	
			ACTION (AFTER, ATOMIC)	location penalty charge == true	WRITE_DATA	ARCA19	
			ACTION (AFTER, ATOMIC)	late return penalty charge == true	WRITE_DATA	ARCA20	
			ACTION (AFTER, ATOMIC)	car drop-off of [rental] must be accepted	C-ACT	ARCA21	
		else ACTION (FIRST, ATOMIC)	car drop-off of [rental] must be rejected	C-ACT	ARCE04(ARCA22)		

Figure 51: Action Rule example following the new syntax

This rule already follows the new syntax proposed and is the result of several iterations of: (1) devising new meta-model constructs for DEMO's Action

Model and (2) evaluating their applicability and comprehensiveness by instantiating all the action rules of the EU-rent case. For step 1 are created instances of UEAOM classes OAK (Organization Artifact Kind), OAKRK (Organization Artifact Kind Relation Kind) and OAKEK (Organization Artifact Kind Element Kind), thus, specifying the meta-model. For step 2 are created instances of UEAOM classes OA, OAR and OAE, that is, creating an Action Model following the specified meta-model.

For this reasoning was used the action rule depicted on Figure 51 .This rule example shows the necessary elements for BPMN specification. The BPMN process based on the action rule regarding the transaction T04 – car drop-off that results from the conversion rules – shown in Figure 52 – depicts the steps of the respective transaction and different kinds of infological and datalogical actions while still being totally abstracted from the implementation. The already mentioned design science research cycle of creation of, both the UEAOM instantiation for the meta-model specification and the instantiation for the full action rule specification of the EU-rent case, continued, until an instantiation gathered in a comprehensive way all the necessary information to have a real-life BPMN process fully specified, excluding implementation details. The full BPMN process of EU-Rent organization can be seen on Appendix C.2-DEMO models into UEAOM based BPMN workflow.

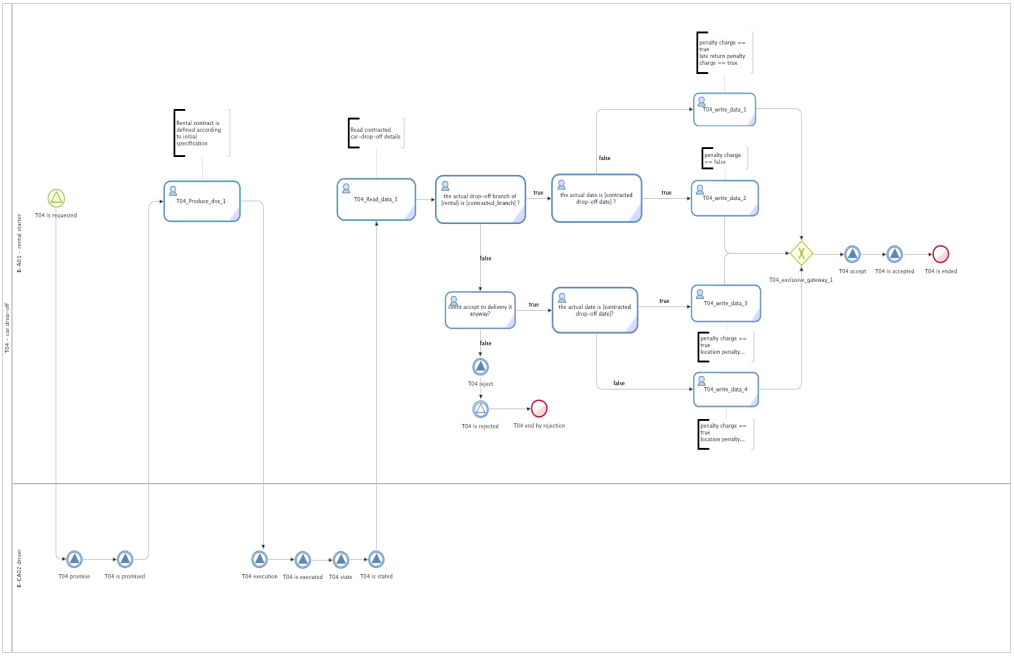


Figure 52: BPMN diagram for Transaction T04 - car drop-off

In the BPMN process example depicted on Figure 52 is find all needed information, namely, all relevant ontological, infological and datalogical steps

regarding the process of dropping the car at a branch. Figure 53 shows the UEAOM instantiation that consists in the new Action Meta-Model proposed and the explanation of each of its elements is now due. Due to the UEAOM following the AOM pattern and also the type square pattern, the explanation that follows could appear confusing, having too many instantiations. But the reader just needs to keep in mind that these patterns provide immense power of adaptability to systems in runtime due to the fact that instances of classes of an AOM may be themselves types or "classes" and that instances of AOM classes may, in turn, have instance kind relationships between them. The instantiation of all the EU-Rent rules can be seen on Appendix C.3-EU-Rent action rules UEAOM instantiation .

Organization Artifact Kind	**Action Rule**	
OAK Relation Kind	**executing actor**	
OAK Element Kind	action rule id	<string>
Organization Artifact Kind	**AR-Component-WHEN**	
OAK Relation Kind	**when component part of action rule**	
OAK Element Kind	arcw-id	<string>
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**	
OAK Relation Kind	**enacting transaction component part of when component**	
OAK Relation Kind	**enacting transaction**	
OAK Element Kind	enacting transaction C-fact	<requested, promised, stated, accepted, revoke request requested, revoke resquet allowed, revoke request refused, revoke promise requested, revoke promise allowed, revoke promise refused, revoke statement requested, revoke statement allowed, revoke statement refused, revoke acceptance requested, revoke acceptance allowed, revoke acceptance refused, rejected, declined>
OAK Element Kind	arcet-id	<string>
Organization Artifact Kind	**AR-Component-CONDITION**	
OAK Relation Kind	**subcondition of condition**	
OAK Element Kind	condition type	<AND, OR, NOT, EXPRESSION>
OAK Element Kind	expression	<string>
OAK Element Kind	arcc-id	<string>
Organization Artifact Kind	**AR-Component-IF**	
OAK Relation Kind	**flow component of action**	
OAK Relation Kind	**evaluated condition**	
OAK Element Kind	arci-id	<string>
Organization Artifact Kind	**AR-Component-FOREACH**	
OAK Relation Kind	**flow component of action**	
OAK Relation Kind	**type of element**	
OAK Element Kind	arcf-id	<string>
Organization Artifact Kind	**AR-Component-WHILE**	
OAK Relation Kind	**flow component of action**	
OAK Relation Kind	**evaluated condition**	
OAK Element Kind	arcwhile-id	<string>
Organization Artifact Kind	**AR-Component-THEN**	
OAK Relation Kind	**then component part of if component**	
OAK Element Kind	ar-ct-id	<string>
Organization Artifact Kind	**AR-Component-ELSE**	
OAK Relation Kind	**else component part of if component**	
OAK Element Kind	arce-id	<string>
Organization Artifact Kind	**AR-Component-ACTION**	
OAK Relation Kind	**action component part of component**	
OAK Relation Kind	**previous action**	
OAK Relation Kind	**fact creation**	
OAK Element Kind	precedence type	<FIRST, AFTER>
OAK Element Kind	action type	<ATOMIC, BLOCK, FLOW>
OAK Element Kind	atomic action type	<WRITE_DATA, READ_DATA, TRANSMIT_DATA, SPECIFY_DATA, PRODUCE_DATA, PRODUCE_DOC, COPY_DOC, STORE_DOC, GET_DOC, READ_DOC, C-ACT, P-ACT>
OAK Element Kind	action description	<string>
OAK Element Kind	generic requirement	<string>
OAK Element Kind	implementation requirement	<string>
OAK Element Kind	arca-id	<string>

Figure 53: UEAOM based Action Rule Meta-Model specification

OAK – Action Rule

The first instance of the UEAOM class OAK that is needed, is of course the one specifying the *action rule* concept type itself. It has an associated OAKEK

instance for its identification that is called *action rule id*. In the example, an instance of the UEAOM class OA would be action rule *AR05*. Being that the AR05 string or value is an instance of an OAE, itself instance of the OAKEK action rule id. This rule AR05 is, itself, an instance – at model level – of the just before mentioned OAK instance action rule – in turn, at meta-model level. To the reader that had no knowledge of the type square pattern. These several "double instantiations" described in the previous sentences constitute an example of 4 AOM instances/objects, forming such a type square, where, on one side we have a type and a property type (part of the type) and on the other side of the square we have an instance of the type containing a value, itself instance of the respective property.

Next, there was the need to specify which actor role has the responsibility of executing this rule. Thus was needed to specify an instance of class OAKRK, that was called *executing actor*, relating the action rule OAK with the actor role OAK. In the example, an instance of an OAR that would be an instance at model level of the OAKRK executing actor (in turn, at meta-model level) would be the the OAR relating AR05 with actor role A01, named rental starter.

OAK - AR-Component-When

Was also needed to specify the fact that triggers an action rule. For that was specified the OAK *AR-Component-When*, also needing an identifier, specified as the OAKEK *arcw-id*. In the example instance is ARCW01. Next, there was the need to specify to which action rule this AR-Component-When is related to. Thus the specification of the following OAKRK: *when component part of action rule*. An instance of an OAR in the example would be the one relating ARCW01 to AR05.

OAK – AR-Component-Enacting-Transaction

The OAK *AR-Component-Enacting-Transaction* serves to specify facts that trigger an action rule. In the example, an OA instance that is an instance of this OAK instance is: ARCET01. This complexity is needed as an action rule can be triggered by one or more c-facts of different transactions. The OAKRK *enacting transaction component part of when component* serves to relate the previous OAK with the *when component* OAK. In the example, an instance of an OAR instance of this OAKRK would be the OAR relating ARCET01 with ARCW01. The OAK Relation Kind *"enacting transaction"* specifies the relation between the OAK *AR-Component-Enacting-Transaction* and the OAK transaction. In the example, an instance of an OAR instance of this OAKRK would be the OAR relating ARCET01 with transaction T04 named car drop-off. The OAKEK *"enacting transaction C-Fact"* serves to specify which fact (out of the possible 21 facts of the universal transaction pattern) of the related transaction triggers the execution of this action rule. In the example there was the OAE *stated*. All the above instances of classes OAK, OAKRK and OAKEK are needed to precisely specify, at meta-model level, the structure of the initial part of an action rule. The above mentioned instances of classes OA,

OAR and OAE specify at model level the following part of our example action rule: *"when car drop-off of [rental] is stated*.

Similar reasonings were followed for all other OAKs, OAKRKs and OAKEKs that were specified and presented in Figure 53. Their specification is now justified.

OAK - AR-Component-Condition

There was the need to specify the OAK *condition*. Conditions are evaluated by what is called *Flows* (If, While), explained later. A condition has to be of a certain (OAKEK) *condition type*, namely: *AND*, *OR*, *NOT*, or *EXPRESSION*. In the first three cases the condition is actually a composite condition where a boolean operator is followed by a (OAKRK) sub-condition. In the last case the condition will actually be an atomic condition in the form of a boolean expression. In the example in Figure 51, there is an instance of class OA, the atomic condition: ARCC01 with expression *the actual drop-off branch of [rental] is [contracted_branch]*. In the case of composite conditions (in fact a tree of conditions) each non-root condition needs to specify that they are *sub-condition of condition*. Imagining a NOT condition with the previous example being the expression, there would be an OAR specifying that ARCC02 is a sub-condition of ARCC01, where the condition type OAE associated with the OA ARSC02 would be: NOT. Having the conditions specified was needed to specify in which context they are evaluated.

OAK – AR-Component-IF/While/Foreach/Then/Else

This happens in two kinds of *flow* component. Both the (OAK) *if* and the (OAK) *while* components evaluate a condition specified by the (OAKRK) *evaluated condition*. It is specified to which action rule they belong thanks to OAKRK *flow component of action.* In the example it would have the OAR: ARCIF01 evaluating the condition ARCC01. The *foreach* flow has an action for each of its *type of element*. An if component is composed by its respective (OAK) *then* and (OAK) *else* components. In the example: the OAK ARCT01 being the *then component part of if component* ARCIF01 and the OAK ARCE01 as the *else component part of if component* ARCIF01.

OAK – AR-Component-Action

Finally there was the need to specify actions of the action rule itself, thus the OAK AR-Component-Action. Actions always belong to some component, relation specified by OAKRK *action component part of component*. In the example: There is an OAR specifying the OA ARCA01 as being the *action component part of component* OA ARCW01 (the when component of action rule AR05). An action component can be followed by another action and such precedences are specified by the OAKRK *previous action*. By this manner the action can be "programmed" sequentially. For instance and based on the example: The OA ARCA05 that has as *previous action* the OA ARCA04.

When an actor perform an action, some fact can be created and that relation is specified by the OAKRK *fact creation*. Since actions can proceed other actions or may be the first action for a certain component, there was the need to be able to specify this differentiation. Thus it was specified the OAKEK *precedence type* where the allowed values for the OAE are strings *FIRST* or *AFTER* . To specify if the action component in question is an atomic action or a block of actions or a flow there was the OAKEK *action type*. In the example: OA ARCA04 is the FIRST action of the then component and is of type BLOCK, that is, it just specifies that there will be a block of actions executed sequentially. The OA ARCA05 is the FIRST ATOMIC action of the action block ARCA04. An atomic action can be of many types and that is specified by the OAKEK *atomic action type*. As seen in Figure 53 there are datalogical, infological and ontological acts as options. In the example: the OA ARCA05 has its OAE with value *WRITE_DATA*. OAKEK *action description* serves to specify/describe in a formal/informal way the atomic action in question. In the example there is the OAE *penalty charge == false*. An action can also have specific requirements, important regarding the implementation but still abstracted somehow from the final implementation, e.g. Rent a car contracts can be written by hand or by computer. It will exist the respective task regarding that action, however the manner how the organization decides to finally implement it can only be decided at the very end of the implementation by whom is implementing it. Along with the OAKEKs *generic requirement* and *implementation requirement* that serve the same reason. As an example of a generic requirement a certain action of type PRODUCE_DATA can be specified as “is mandatory” which means that the actor has to really get or produce such data from somewhere. An example of an implementation requirement would be “obtain record from the government driving license system's web service”.

An OFD (the DEMO option for specifying meta-models) can be produced for another alternative view (other than the UEAOM instantiation) of the newly proposed meta-model for DEMO Action Rules.

All the re-written EU-Rent action rules regarding the new action rule can be seen on Appendix C.1-EU-Rent action rules new syntax. Along with that, the full BPMN process regarding all the transactions of the use case example, result of the appliance of the conversion rules are depicted on Appendix C.2 - DEMO models into UEAOM based BPMN workflow.

Action Rule syntax

For the representation of the new syntax, the BNF [36] (Backus Naur Form) notation is used.

In computer science, BNF is often used to describe syntax of languages used in computing, as programming languages, documents formats, instruction sets and communication protocols [36]. Due to be widely used to this kind of specifications, BNF is used to describe the Action rule new syntax.

Table 8 describes the new syntax. This new action rule syntax aims to provide a solid way of having a real overview on how an organization have its processes, the involvement of the employees as well as their role in the organization. The new syntax is based on the previous known syntax and in order to make it simple and clear it was mainly reduced to actions, conditions and its respective evaluations. Everything on an organization gets going with someone's action and involvement on it.

A simple and direct explanation of the action rule new syntax is now due.

Table 8: Action Rule new syntax proposal

<When> : : =	<Transaction kind name> "of" <Object Identifier> "is" <C-Fact> | <P-Fact> <Action> | <Flow>
<Transaction kind name> : : =	<String>
<Object Identifier> : : =	"[" <String> "]"
<P-Fact> : : =	"produced"
<C-Fact> : : =	"requested" | "promised" | "stated" | "accepted" | "revoke request requested" | "revoke resquet allowed" | "revoke request refused" | "revoke promise requested" | "revoke promise allowed" | "revoke promise refused" | "revoke statement requested" | "revoke statement allowed" | "revoke statement refused" | "revoke acceptance requested" | "revoke acceptance allowed" | "revoke acceptance refused" | "rejected" | "declined"
<C-Act> : : =	<Transaction kind name> "of" <Object Identifier> "must be" <C-Fact>
<P-Act> : : =	<Transaction kind name> "of" <Object Identifier> "must be" <P-Fact>
<Flow> : : =	<Foreach> | <If> | <While>
<Action> : : =	<Atomic action> | <Block> | <Flow>
<Atomic action> : : =	<Action type> <Action description> | <Generic requirement> | <Implementation requirement>
<Action description> : : =	<String>
<Generic requirement> : : =	<String>
<Implementation requirement> : : =	<String>
<Action type> : : =	"Write_Data" | "Read_Data" | "Transmit_Data" | "Specify_Data" | "Produce_Data" | "Produce_Doc" | "Copy_Doc" | "Store_Doc" | "Get_Doc" | "Read_Doc" | <C-Act> | <P-Act>
<Block> : : =	<Action> { <Action> }
<Condition> : : =	<Condition type> { <Condition> } | <Expression>
<Condition type> : : =	"And" | "Or" | "Not" | <Expression>
<Expression> : : =	<String>
<If> : : =	"If" <Condition> "then" <Action> | <Flow> "else" <Action> | <Flow>
<While> : : =	"While" <Condition> <Action> | <Flow>
<Foreach> : : =	"Foreach" <type of element> <Action> | <Flow>

A transaction is composed by several states, the Order-Phase, the Execution-phase and the result-phase[14]. These phases and regarding the action rules, start with a "*When*" component. When a *transaction kind* of some object (ex: a rental) is in a *C-Fact* (requested, promised, stated or Accepted) or a *P-Fact* (executed/produced) state an *Action* or a *Flow* can proceed. *Flows* and *Actions* are basically the main and the most used components on this new syntax. The *Action* can be an *atomic action* (Unique action) a *Block of actions* and a *Flow*. A *Block* of actions is seen as a set of interrelated Actions. A *Flow* can be an If, a Foreach or a While, with the responsible of verifying its respective *condition*. Depending on the result of those conditions – normally true or false - the

14 More details about the Operation Axiom can be seen on Sub-chapter 2.1.2-The PSI-Theory

transaction can proceed by different flow paths.

The most important gain and contribution of this new syntax is definitely its versatility, the gathering of the really important information needed for the conversion and the possibility of controlling the flow of the process, including actions and their role on the transaction. All of this advantages, but even maintaining the implementation independence, the main characteristic of applying DEMO methodology into organizations.

5.4 BPMN generation and implementation example

A complete transaction, and according to the standard transaction pattern[15], is composed by all its C-Acts/C-Facts and P-Act/P-Fact (request, promise, execution, state, accept), possible conditions and its respective evaluations. Some of its construction C-Acts/C-Facts are tacitly, however they must be part of the transaction construction in this particular situation due to the conversion. Depending on the agendum of its respective actor the acts/facts can be all over the available and constructed action rules. It will always depend on the actor agendum across all transactions.

In order to construct the compliant BPMN model, and in this case, the process regarding the T05 transaction named *Car Drop-off* a gathering of all the transaction states among all the action rules available is necessary, so a complete transaction pattern can actually be constructed.

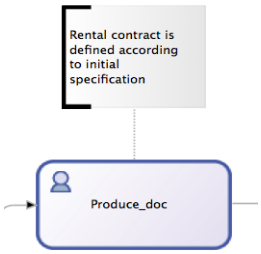
The *car drop-off of [rental]* is requested when the *rental start of [rental]* is promised, so that a throw event from T01 named T04 - request is caught by the start event *T04 - is requested* part of the T04 process. The consequent C-Act/C-Fact promise happens naturally (tacitly) since the standard transaction pattern is used in the example. A throw event for both promise C-Act and C-Fact is used. An action must be performed as a task, so the creation of a drop-off contract must be represented on the BPMN as a task. On the contracted date and location, the renter must deliver the car, so the real execution/production of the transaction. The respective Production-Act and Production-Fact of the *car drop-off* are converted as throw events. After the delivery, the rental contract must be read by the receiver (receptionist) of the car so the conditions can be correctly evaluated. Continuing the same reasoning, the action represented on the action rule must be converted as tasks on the BPMN so all the evaluations must be evaluated on a BPMN task. The conditions evaluations and depending on the implementation details can be made by a human or can be treated by the information system. Being checked by the human, one simply checks according the previously and already read contracted details and if it matches the actual conditions. Is the actual branch the contracted delivery branch? Client accepted it to delivery anyway? Proceeding with a “yes” or a

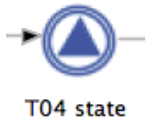
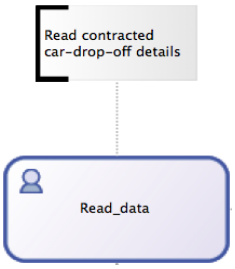
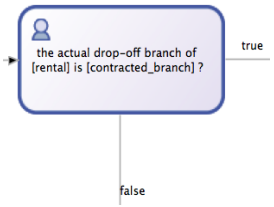
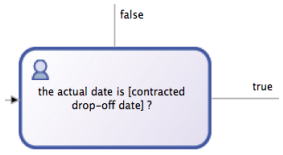
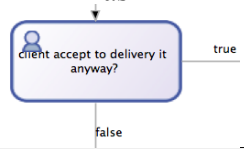
15 More detail about the standard transaction pattern can also be seen on Sub-chapter 2.1.2- The PSI-Theory

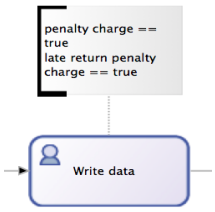
"no", chose by the receptionist for each of the condition evaluation the final result of the transaction is reached by deciding if the client really delivers the car or not so the *car drop-off of [rental] must be accepted* fact must be created.

On Table 9 the applied conversion rules are thoroughly described so each of it is actually evaluated on the BPMN construction.

Table 9: DEMO to BPMN Conversion rules application

Action rule new syntax component			**BPMN notation**
Action	Car drop-off of [rental] must be requested	C-Act	Throw event 
Tacit	Car drop-off of [rental] is requested	C-Fact	Catch Start Event 
Tacit	Car drop-off of [rental] must be promised	C-Act	Throw event 
When	Car drop-off of [rental] is promised	C-Fact	Throw event 
Action	Rental contract is defined according to initial specification	Produce document	Task 

Tacit	Car drop-off of [rental] execution Car drop-off of [rental] is executed	P-Act P-Fact	Throw events 
Tacit	Car drop-off of [rental] must be stated	C-Act	Throw event 
When	Car drop-off of [rental] is stated	C-Fact	Throw event 
Action	Read contracted car drop-off details	Read Data	Task 
Condition	the actual drop-off branch of [rental] is [contracted_branch]	Fact Evaluation	Task 
Condition	the actual date is [contracted drop-off date]	Fact Evaluation	Task 
Condition	client accept to delivery it anyway	Evaluation	Task 

Action	penalty charge == true late return penalty charge == true location penalty charge == true	Write data	Task 
Action	Car drop-off of [rental] must be accepted	C-Act	Throw event 
Action	Car drop-off of [rental] is accepted	C-Fact	Throw event 
Action	Car drop-off of [rental] must be rejected	C-Act	Throw event 
Action	Car drop-off of [rental] is rejected	C-Fact	Catch event 

With the application of the suggested conversion rules[16] the BPMN process based on DEMO models of the organization is depicted on Figure 52. The implementation details (Data base connections, human tasks, service tasks) are responsibility of the final implementation manager. Even with some requirements that are already included in the specification of the action rule construction, it continues to be implementation independent, so the necessity of some kind of human hand readjustments to the final converted model are actually needed.

5.5 Conclusion

When one really starts to specify details, still at infological and datalogical

16 Conversion rules can be seen on Sub-chapter 4.4-Conversion rules between models

levels things can get really complex, as it was seen when some realistic detail was added to the action rules. By looking at the full model of EU-Rent [33], one can see that it is only on the action rule that handles the promise of the transaction rental end, and that the payment transaction is requested (possibly with the fines). Imagining that the renter would drop off the car by mistake in a branch different from the contracted one, after dropping off the car at the garage he or she would get quite a surprise at the branch desk when having to pay the fine. It makes much more sense that the action rule that handles the state c-fact of the transaction car drop-off evaluates if the drop-off branch is correct and already informs the renter of the fine he would have to pay if he wishes to proceed, to give him a chance of not proceeding and maybe leave this branch and deliver the car on the correct branch. Just this example shows that, while modeling DEMO transactions, one should already have in mind implementation issues that will affect transaction design. Depending if (1) there is a garage attendant and then the rental desk handles the payment or if (2) there is just the garage attendant himself which takes care of both the car drop-off and penalty payments; this will have a profound impact on the design of the action rules and maybe even on transaction design itself. In this later case, one could “fuse” current transactions car drop-off and rental end. In the more complete action rule presented, is possible to check that situation, an apparently simple rule was in fact “forgetting” lots of complexity of conditions to be verified and many original facts creation (e.g., the flags regarding the fines) that are very relevant actions to be correctly and comprehensively implemented in a BPMN flow in the correct spot of the flow.

CHAPTER 6

Conclusion

A long path and a kind of forth and coming events were taken until the successfully conclusion of this graduation project was reached. This final chapter aims to discuss the entire graduation project and the relevant issues of it for possible future work.

It includes, in Sub-chapter 6.1-Project results , the final conclusions about the approach and the obtained results. In Sub-chapter 6.2-Future work, is where some ideas for gathering information of this project and applying it on some new interesting concepts were discussed, along with some still possible improvements.

6.1 Project results

This dissertation project started off with the purpose of automatically generating runnable BPMN models from DEMO models. The idea was to take advantage of a well defined methodology as is the DEMO, and convert it, into a runnable workflow model. Being the BPMN as a standard of the business process management, was concluded that it would be the most suitable notation for the required job.

For starters, was definitely solved the problem of lack of semantics (for this particular case) in BPMN, by selecting a few BPMN concepts (out of the many ambiguous ones available) and assigning to them clear and precise semantics thanks to the 1 to 1 conversion rules from DEMO to BPMN that was proposed. But in the middle of the process was found out another problem: that the main information source in DEMO to generate BPMN flows – the Action Rules

Specification of the Action Model – was not precise enough nor had clear semantics itself. Then was took the endeavor of, following Design Science Research tenets, solving that problem by applying the Universal Enterprise Adaptive Object Model to specify a more complete and comprehensive Meta-Model, i.e., abstract syntax, for the Action Rules. After several instantiations of the main generated ideas and evaluation of their applicability by instantiating all EU-rent case action rules with each new version of the Meta-Model, the idea kept on improving to the stage as is presented on this project. Some very relevant contributions of the new syntax were the finding of the primitives for what could be called the *organization programming language*, still at an implementation abstracted level, in a way that can be produced much more comprehensive BPMN models, almost ready to be runnable.

Flexibility and interoperability will definitely be added to the organizations that make use of the DEMO methodology on its definition processes or even for the ones already using it. This contributions adds lots of potentially to its actual transactions with the possibility of converting them into BPMN runnable processes.

6.2 Future work

Based on this graduation project, future work or even some improvements to what is already made is possible.

The current version of the syntax has several aspects to improve still. Namely when exists an atomic action that is a C-ACT, there must be an additional OAKEK to specify to which transaction this C-ACT corresponds, so that the workflow engine can throw the right signal to activate the right BPMN pool. In fact one of the next lines of future work, besides polishing up missing details in the new syntax, is the creation of a parser that takes as an input action rules following the proposed syntax and outputs a runnable BPMN workflow that can be automatically imported to a well-known Open Source Workflow System where can be easily implement (or connect to) the needed database tables, queries and web forms for data production/input, following the requirements formally or informally specified in each atomic action of the action rules.

There is also another small contribution to the formal specification of the Universal Transaction Pattern. Since all coordination acts need to have different names, and for instance, revoking a request is different from revoking a promise, the allow and refuse acts should also be differentiated according to if they correspond to allowing the revoke of a request or of a promise. Thus all the new names of c-acts that can be found on the new syntax and the pattern should have also the names of these acts changed accordingly.

Bibliography

[1] S. Dalal and D. R. S. Chhillar, “Case Studies of Most Common and Severe Types of Software System Failure,” *International Journal of Advanced Research in Computer Science and Software Engineering*, p. 7, Aug-2012.

[2] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz, “What We Have Learned About Fighting Defects,” in *in Proceedings of 8th International Software Metrics Symposium*, 2002, pp. 249–258.

[3] A. Zeller and R. Hildebrandt, “Simplifying and Isolating Failure–Inducing Input,” *IEEE Trans. Onsoftware Eng.*, vol. 28, no. 2, pp. 2–15, Feb. 2002.

[4] D. Sandeep and R. Chhillar, “Role of fault Reporting in Existing Software Industry,” vol. 4, no. 7, pp. 3–15, Jul. 2012.

[5] Y. Wang, “Transformation of DEMO models into exchangeable format,” Delft University of Technology, Netherlands, 2009.

[6] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design science in information systems research,” *Manag. Inf. Syst. Q.*, vol. 28, no. 1, pp. 75–106, 2004.

[7] A. R. Hevner, “A Three Cycle View of Design Science Research,” *Scand. J. Inf. Syst.*, vol. 19, no. 2, Jan. 2007.

[8] J. L. G. Dietz, “A World Ontology Specification Language,” in *On the Move to Meaningful Internet Systems 2005: OTM Workshops*, S. B. / Heidelberg, Ed. 2005, pp. 688–699.

[9] T. R. Gruber, “What is an Ontology?” [Online]. Available: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html. [Accessed: 13-Dec-2013].

[10] “Ontology,” *Wikipedia, the free encyclopedia*. 31-Jul-2013.

[11] J. L. G. Dietz, *Enterprise ontology theory and methodology*. Berlin; New York: Springer, 2006.

[12] J. D. Haan, “Modeling an organization using Enterprise Ontology,”

The Enterprise Architect, 10-Oct-2009. [Online]. Available: http://www.theenterprisearchitect.eu/archive/2009/10/10/modeling-an-organization-using-enterprise-ontology. [Accessed: 01-Aug-2013].

[13] J. Mulder, "Rapid Enterprise Design." Delf University of Technology, 2006.

[14] "BPM Tutorial - TechTarget," *TechTarget*. [Online]. Available: http://searchsoa.techtarget.com/tutorial/BPM-Tutorial. [Accessed: 23-Jul-2013].

[15] Jeston, *Business Process Management (06) by Jeston, John - Nelis, Johan [Hardcover (2006)]*. ButerworthHeineman, Hardcover.

[16] M. Walton, *The Deming Management Method*. Penguin, 1986.

[17] R. T. Burlton, *Business process management: profiting from process*. Indianapolis, Ind.: Sams, 2001.

[18] "What is BPM? An introduction to Business Process Management," *Winscribe*. [Online]. Available: http://www.winscribe.com/business-process-management/what-is-bpm. [Accessed: 30-Jul-2013].

[19] A. Skjekkeland, "What is BPM?," 12-Feb-2009.

[20] "Object Management Group," *Wikipedia, the free encyclopedia*. 12-Jul-2013.

[21] J. Arlow and I. Neustadt, *Introduction To BPMN 2*, vol. 1. Mountain Way Limited, 2012.

[22] G. Polančič and T. Rozman, "BPMN 1.0 Poster," *ITPoster.net*. [Online]. Available: http://www.itposter.net/itPosters/bpmn/bpmn.htm. [Accessed: 25-Jul-2013].

[23] "BPMN 2.0 Poster – www.bpmb.de." [Online]. Available: http://www.bpmb.de/index.php/BPMNPoster. [Accessed: 25-Jul-2013].

[24] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and Analysis of Business Process Models in BPMN," *Inf Softw Technol*, vol. 50, no. 12, pp. 1281–1294, Nov. 2008.

[25] P. M. Matsumoto, F. F. Correia, J. W. Yoder, E. Guerra, H. S. Ferreira, and A. Aguiar, "AOM Metadata Extension Points," presented at the 18th Conference on Pattern Languages of Programs (PLoP), 2011, p. 16.

[26] J. W. Yoder, F. Balaguer, and R. Johnson, "Architecture and Design of Adaptive Object-Models," 2001, pp. 50–60.

[27] D. Aveiro and D. Pinto, "Universal Enterprise Adaptive Object Model," presented at the KEOD 2013, Vilamoura, Portugal, 2013.

[28] T. Halpin, "Object-Roel Modelling: an overview." 1998.

[29] H. S. Ferreira, F. F. Correia, and L. Welicki, “Patterns for Data and Metadata Evolution in Adaptive Object-models,” in *Proceedings of the 15th Conference on Pattern Languages of Programs*, New York, NY, USA, 2008, pp. 5:1–5:9.

[30] “Colosa, Inc.” [Online]. Available: http://www.processmaker.com/. [Accessed: 28-Feb-2013].

[31] T. Rademakers, “Activiti in action executable business processes in BPMN 2.0,” 2012. [Online]. Available: http://proquest.safaribooksonline.com/?fpi=9781617290121. [Accessed: 29-Jul-2013].

[32] “Bonitasoft | Open Source Workflow & BPM software.” [Online]. Available: http://www.bonitasoft.com/. [Accessed: 28-Jan-2013].

[33] J. L. G. Dietz, “DEMO 3 - Way of Working.” 01-Sep-2009.

[34] J. L. G. Dietz, “DEMO-3 Models and Representations - in www.demo.nl,” 2009.

[35] J. Pombinho, D. Aveiro, and J. Tribolet, “The Role of Value-Oriented IT Demand Management on Business/IT Alignment: The Case of ZON Multimedia,” in *Practice-Driven Research on Enterprise Transformation*, F. Harmsen and H. A. Proper, Eds. Springer Berlin Heidelberg, 2013, pp. 46–60.

[36] “Backus–Naur Form,” *Wikipedia, the free encyclopedia*. 25-Sep-2013.

[37] “Business Process Model and Notation,” *Wikipedia, the free encyclopedia*. 18-Jul-2013.

Appendices

Appendix A - Frameworks installation

A.1 ProcessMaker

The open source version is free to use for unlimited period of time, and the others versions could be tested for a limited period of time, after that a license must be acquired. All the versions can be downloaded directly from its official web site.

Based on the version 2.0, its installation can be made from different ways on different machines or even on different platforms. Could be used an Automatic installation if it done on a Windows Machine or a Manual installation for a Linux/Unix machine.

Figure 54: PM Installation - Select and download

For this purpose, was used a Windows 8 automatic installation of the ProcessMaker Open Source Free Edition.

For a manual installation on a Linux/Unix Machine please refer to

http://wiki.processmaker.com.

The software and its correct version can be acquired on the official web site.

After the download of the correct version, in this case 2.0.45, the file with the name ProcessMaker-2.0.45-Setup.exe will be stored in downloads folder.

Double click on the executable file and the next steps happens as follows:

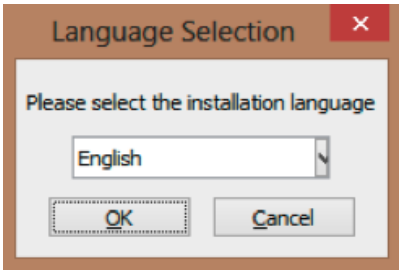


Figure 55: PM Installation - Select the installation language

After selecting the installation language the installation will proceed.

Figure 56: PM Installation - Start installation

Chose the folder to the installed files:

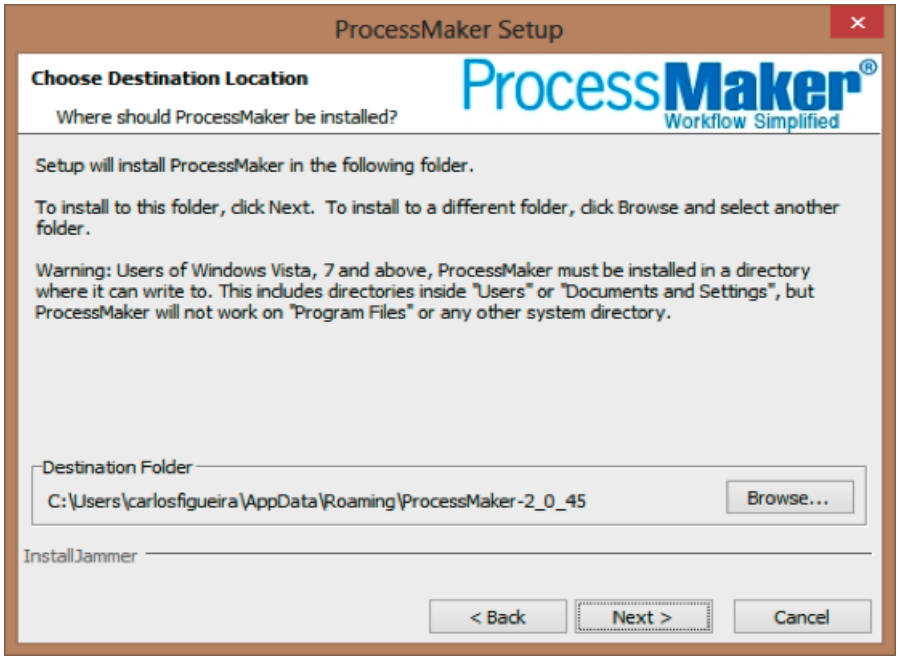


Figure 57: PM Installation - Select installation folder

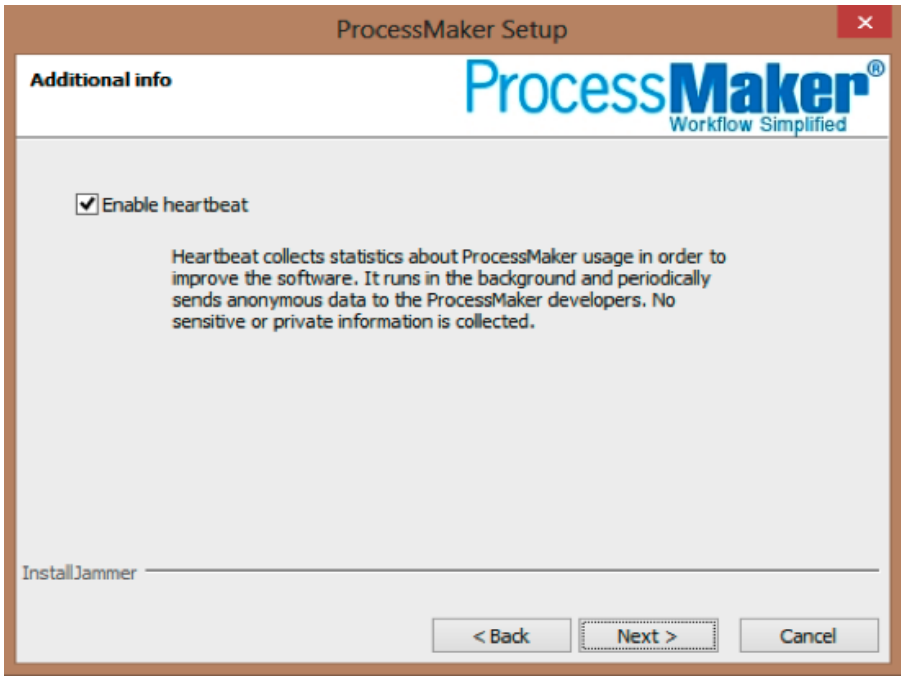


Figure 58: PM Installation - Additional info

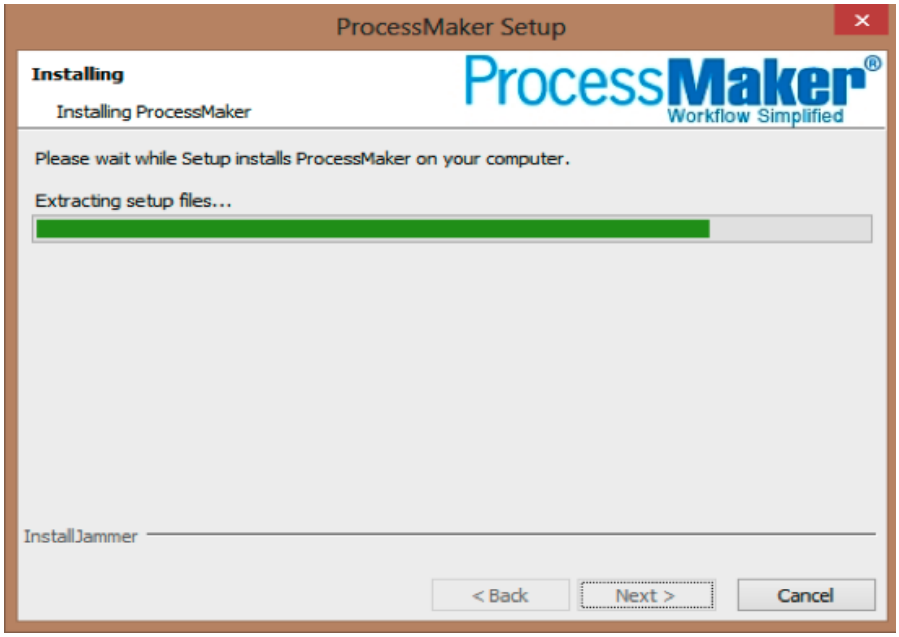


Figure 59: PM Installation - Extract files

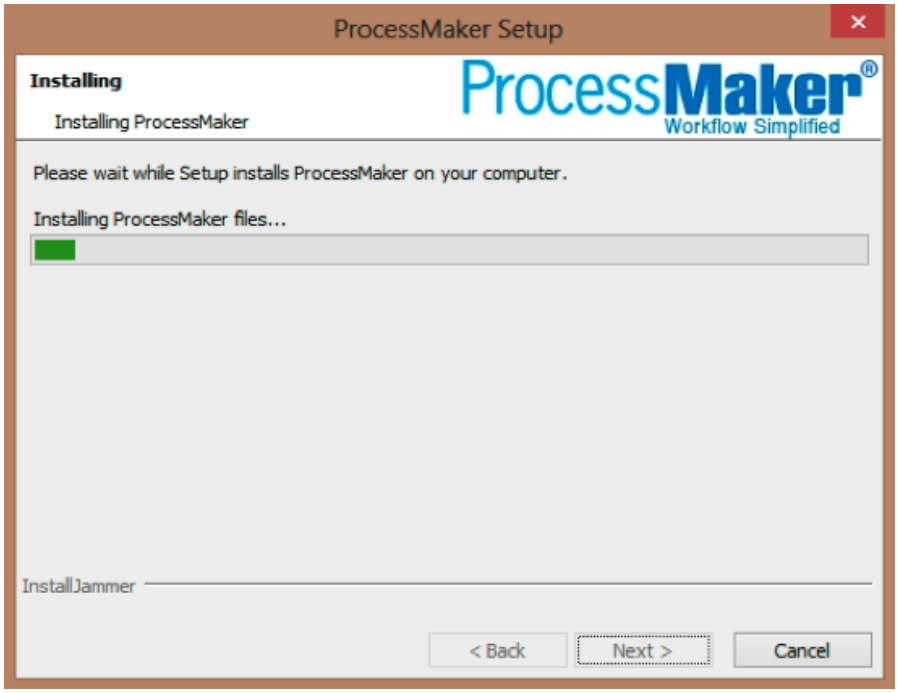


Figure 60: PM Installation - Installing files

Figure 61: PM Installation - Finishing installation

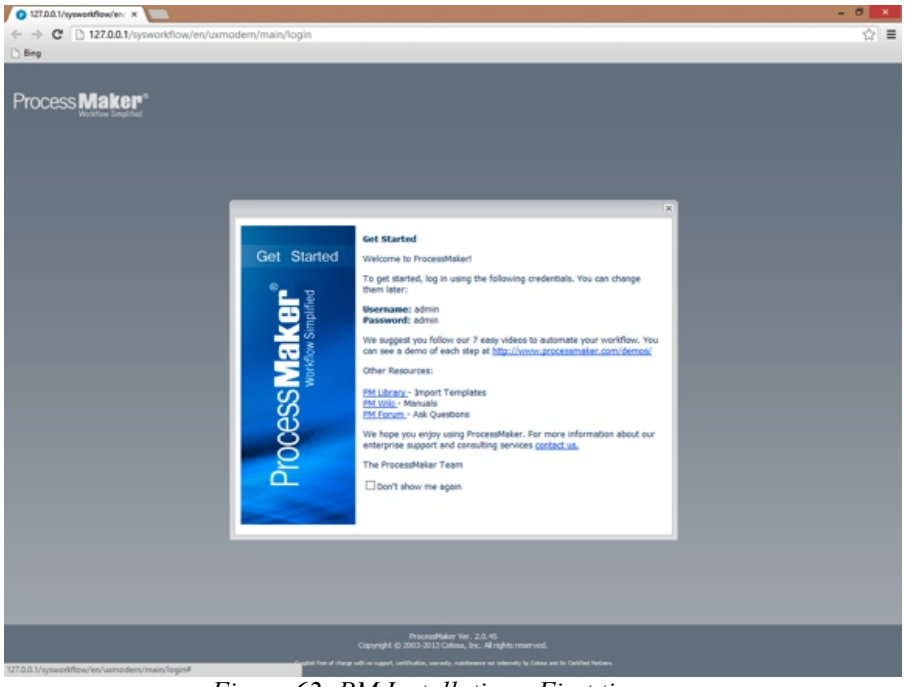


Figure 62: PM Installation - First time run

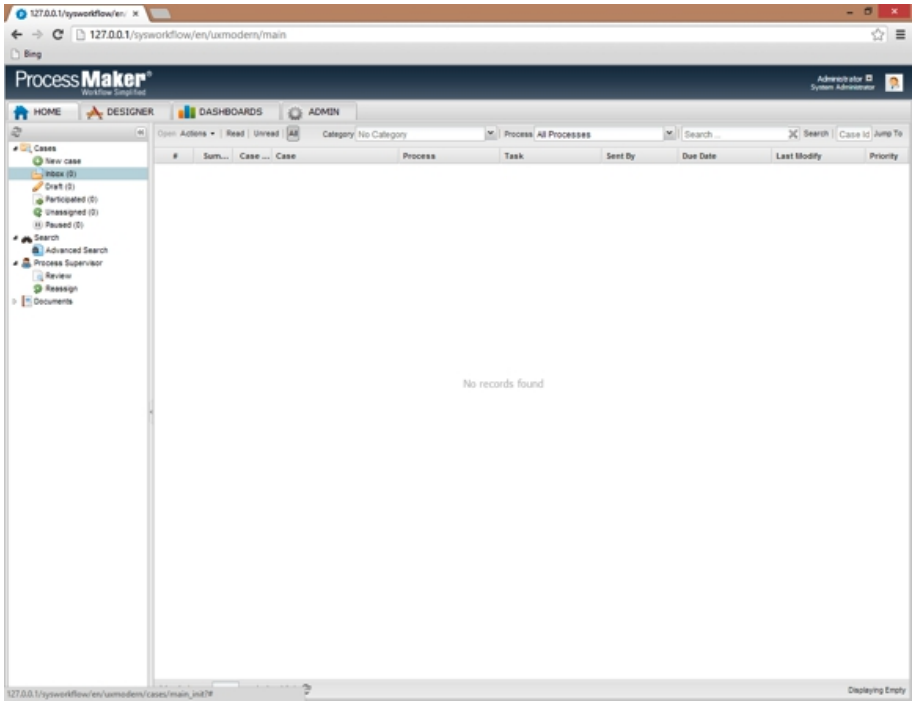


Figure 64: PM Installation - Main dashboard

A.2 Activiti (version 5.11 plus required tools)

One of the good things about Activiti is its user-guide. It is extremely complete and explains in detail what to do in order to have the framework in good working order. It starts with the java runtime machine installation, then the Apache Tomcat to end with the how-to-install the Designer plugin tool on the Eclipse IDE. The installation of all the necessary tools (Java Runtime Machine and Apache Tomcat server) is cross-platform and works the same way on each one of them.

Java Runtime

In order to Activiti works properly, a JRE (Java Runtime Environment) needs to be installed.

The JRE could be downloaded from:

http://www.oracle.com/technetwork/java/javase/downloads/jre7-downloads-1880261.html

Depending on the system, in which JRE is going to be installed, the installation can vary, however the details of each installation can also be found on www.oracle.com.

Apache Tomcat

Along with JRE, an Apache Tomcat server is necessary. The Tomcat could be found on the official website www.tomcat.apache.org.

Activiti Designer Eclipse Plug-in

On the last version of Activiti (5.11) the main added feature was the ability to create and edit the BPMN directly on the Activiti Explorer. However, this ability is not so completed in terms of BPMN edition, as it is in the Eclipse Plug in. That is why the installation of the Activiti Designer Plug in for Eclipse is still quite helpful.

In order to the Eclipse Designer plug-in work properly the Juno or Indigo distribution must be installed.

The following installation instructions are verified on Eclipse Juno and should be similar with the Indigo distribution.

Go to Help menu and select Install New Software.

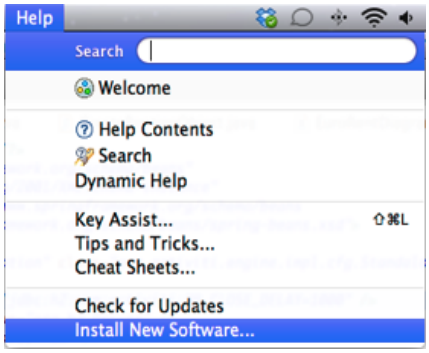


Figure 65: Activiti Installation - Designer plug-in

On the following panel, click *Add* button and fill in the fields with the information:

Name: Activiti BPMN 2.0 designer

Location: http://activiti.org/designer/update/

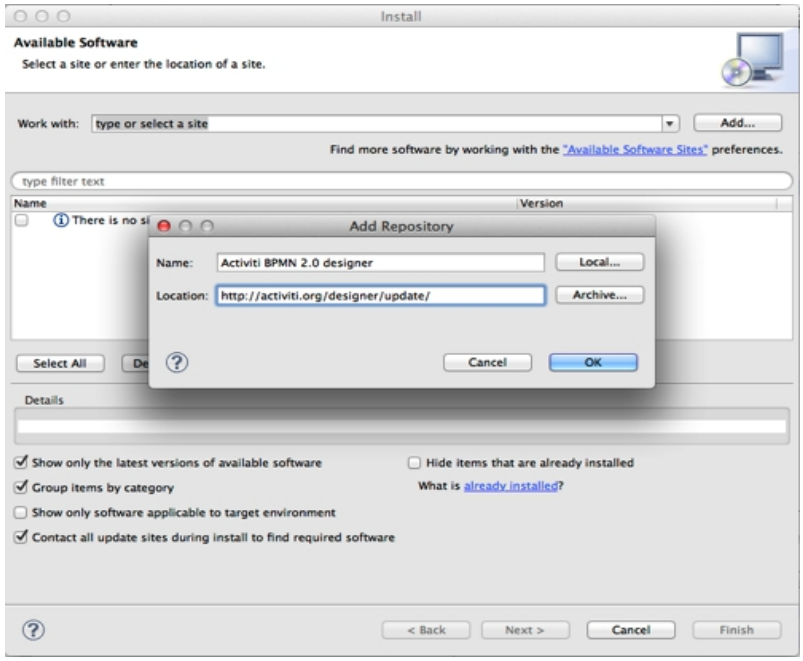


Figure 66: Activiti Installation - Designer plug in - Repository

Make sure the “Contact all updates sites” checkbox is checked.

With this the possibility of creating new Activiti Projects is available.

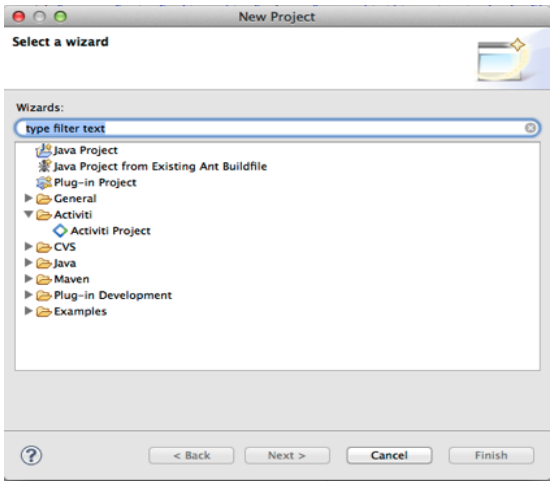


Figure 67: Activiti Installation - Designer plug in - New project

Activiti Explorer

Once in the Activiti website (www.activiti.org), just need to select the Downloads tab and chose the desired version.

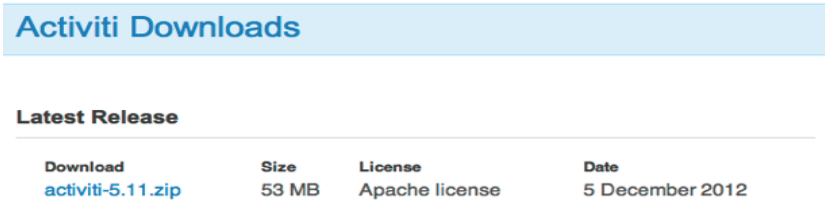


Figure 68: Activiti Installation - Activiti Explorer - Select version

After the download and having all the necessary tools installed, (JRE, Apache Tomcat) one needs to copy the downloaded activity-explorer.war to the webapps directory of Tomcat.

Next step is to start the Tomcat by running the startup.bat or startup.sh scripts in the bin folder of Tomcat and make sure it is running. Pointing the browser to http://localhost:8080

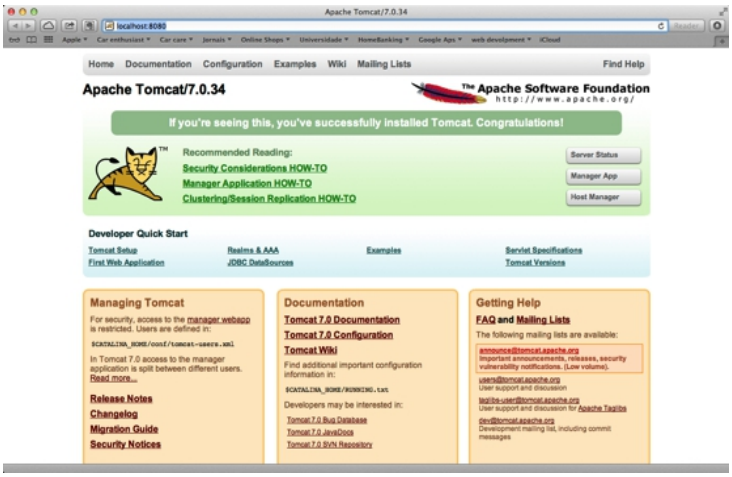


Figure 69: Activiti Installation - Activiti Explorer - TomCat running

The final step is point the browser to the http://localhost:8080/activiti-explorer and log in with the user kermit and password kermit.

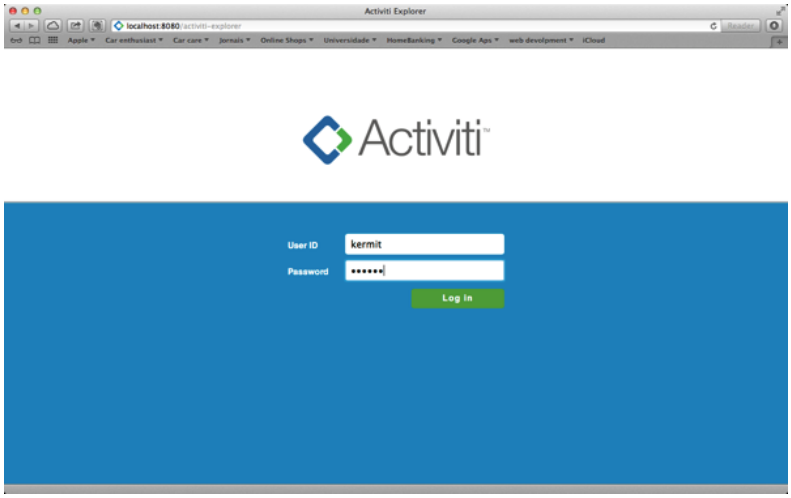


Figure 70: Activiti Installation - Activiti Explorer - Logging in

A.3 BOS (version 5.9.1)

The Open Source Edition is used in this How-to installation in a Unix/Linux Machine, running the Mac OX 10.8 Mountain Lion.

Once in the official web site, one just selects the desired version.

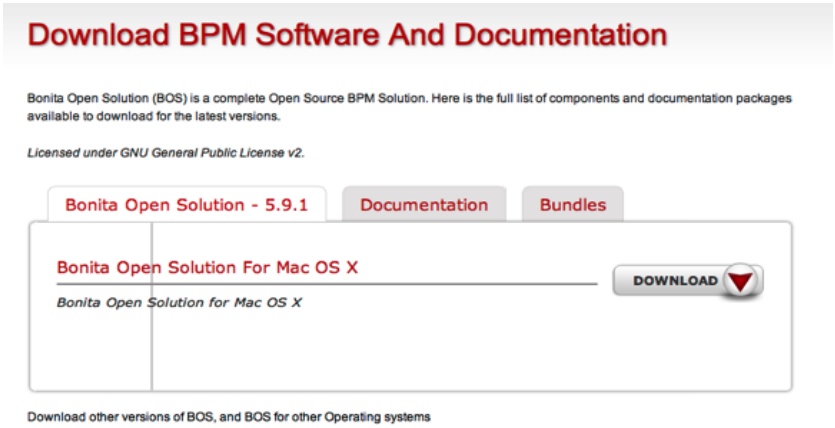


Figure 71: BOS Installation - Selecting version

Selected the desired version, is required to fulfill the registration form, so then, the download can began. The registration is used for end statistics only.

Figure 72: BOS Installation - Registration

Downloaded the latest release of the software, the file BOS-xxx-mac-setup.dmg (xxx means the current version, in this case the 5.9.1 version was downloaded) is added to Downloads Folder.

Once clicked the .dmg file, it will be mounted and the BOS-xxx-setup.app file is shown.

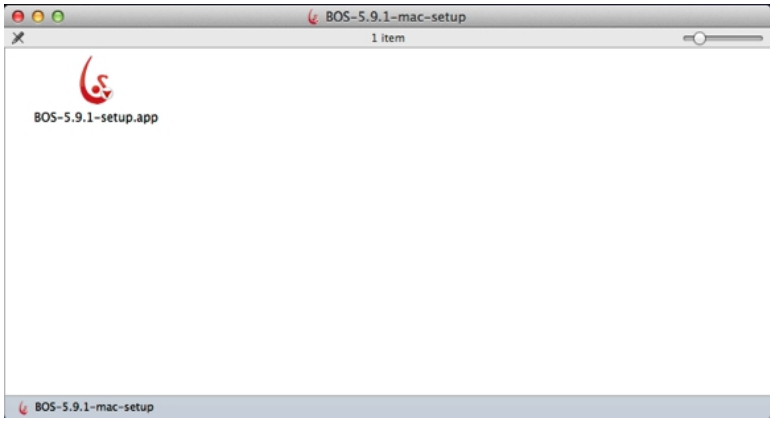


Figure 73: BOS Installation - Setup installation start

Figure 74: BOS Installation - Selecting installation language

Figure 75: BOS Installation - Selecting JVM to use with BOS

Figure 76: BOS Installation - Setup beginning

Figure 77: BOS Installation - License agreement

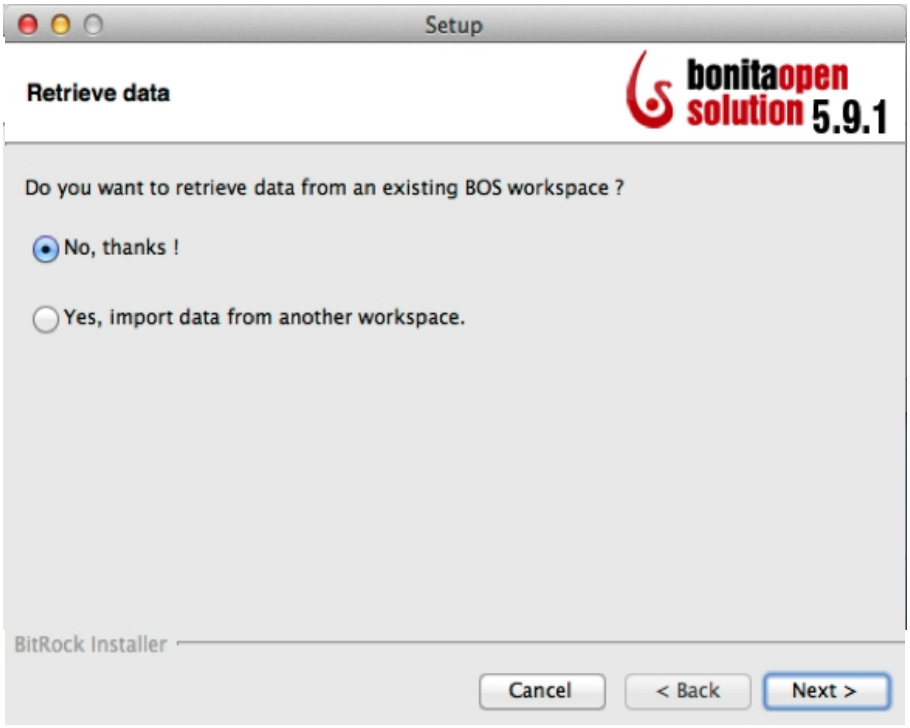


Figure 79: BOS Installation - Retrieve data from others installations

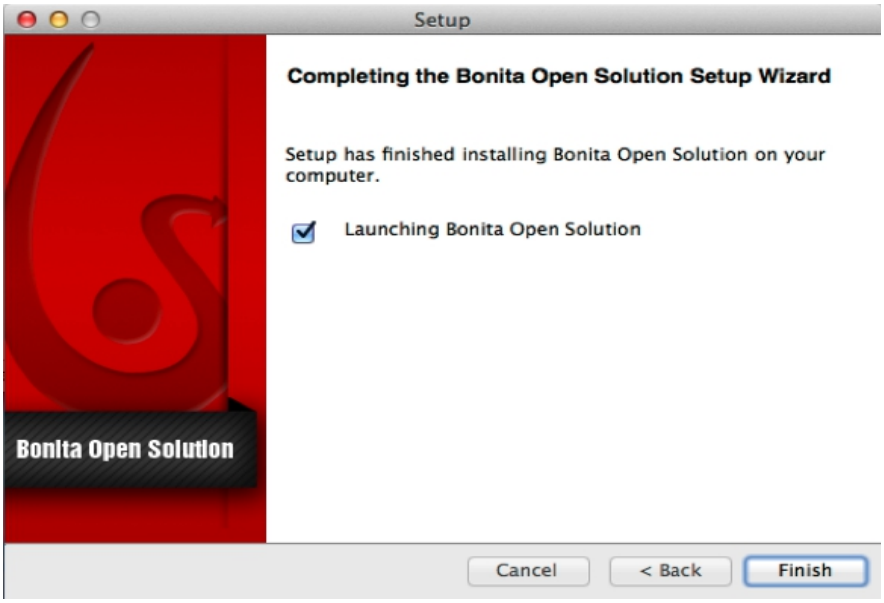


Figure 80: BOS Installation - Installation complete

Figure 81: BOS Installation - BOS initializing

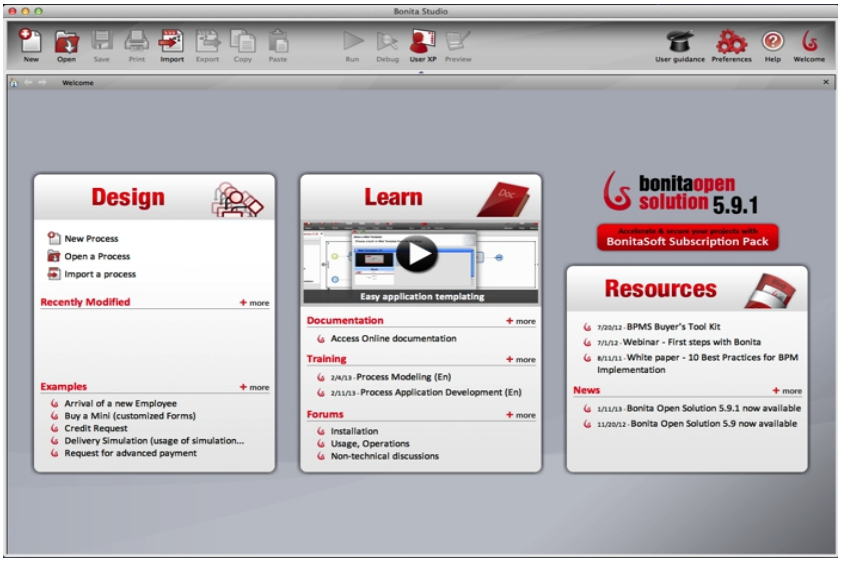


Figure 82: BOS Installation - Main dashboard

Appendix B - Creating BPMN models

B.1 Creating BPMN models on BOS

Once on the main page of the BOS, creating a new BPMN process is quite simple. Just click New Process.

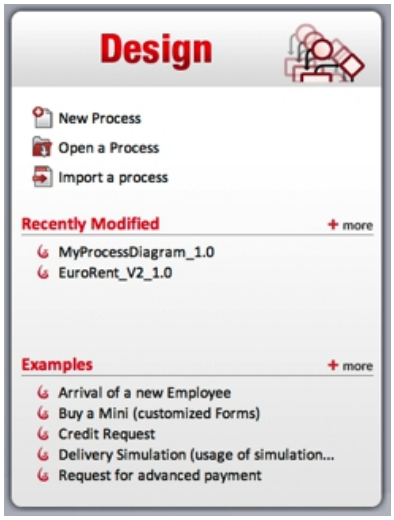


Figure 83: Create BPMN model on BOS - New model

Created a new process, on the general Tab the information about the process, and for instance the name, can be changed.

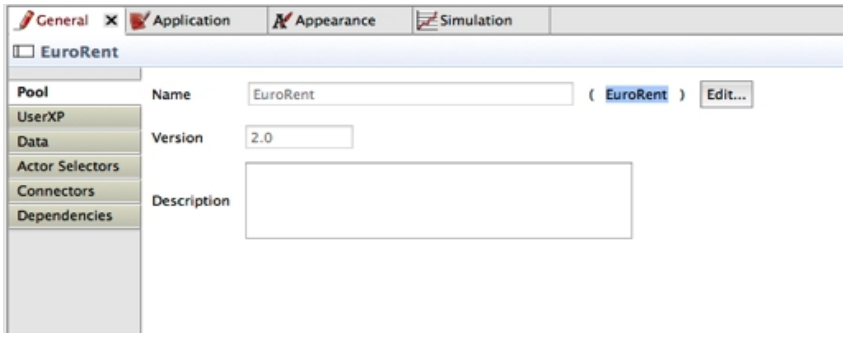


Figure 84: Create BPMN model on BOS - General information

New processes from various types (Abstract, Script, Human and others) can be added using the left menu.

Figure 85: Create BPMN model on BOS - Left menu

Adding forms to a Human process type

After added a Human type process, a form for input data can be added.

The desired process needs to be selected:

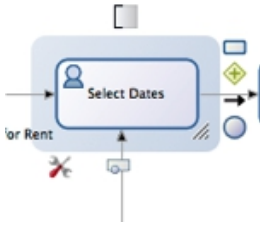


Figure 86: Create BPMN model on BOS - Select the process

Once selected the process the Application Tab must be selected.

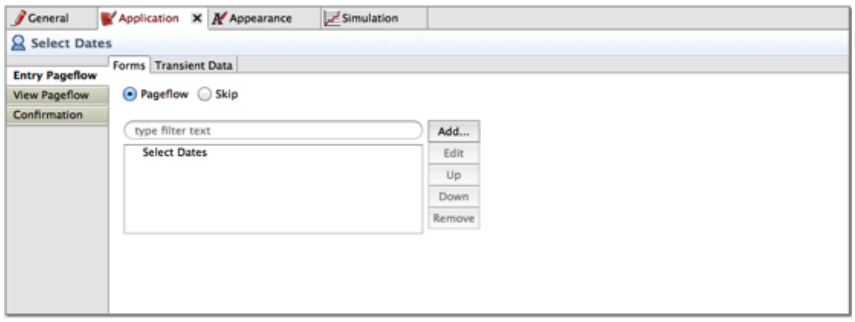


Figure 87: Create BPMN model on BOS - Adding forms 1

Clicking on the Add button a wizard would be shown so the variables for the form could be selected.

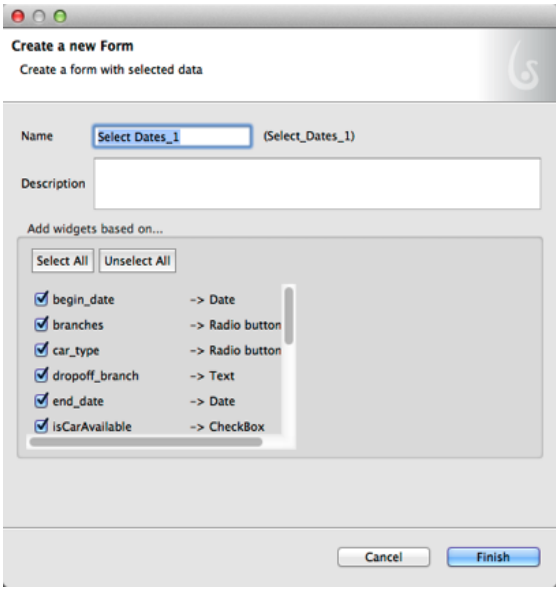


Figure 88: Create BPMN model on BOS - Adding forms 2

After that a tab showing the form options is showed. All the fields of the form and input data can be changed or new ones can be added.

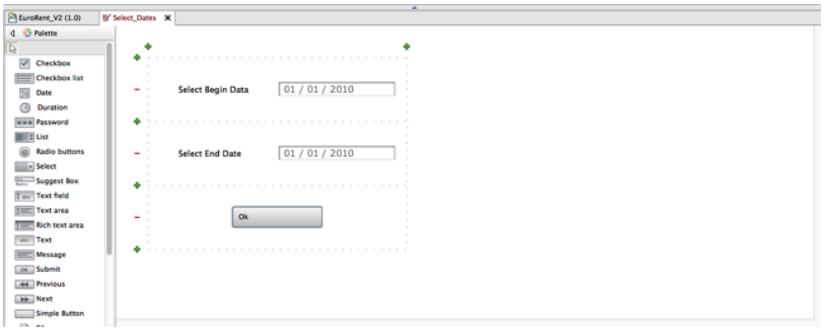


Figure 89: Create BPMN model on BOS - Adding form fields

All done with the configuration a preview of the input form for the desired process could be viewed.

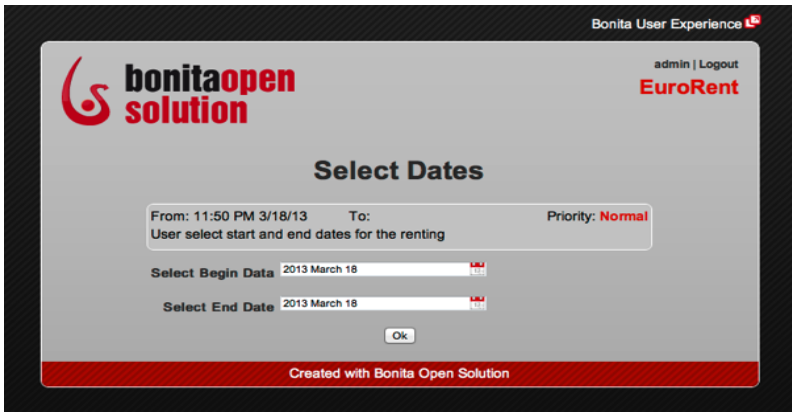


Figure 90: Create BPMN model on BOS - Run the preview form

Adding a MySQL connection

As the Human process type, in this case a Service process type must be chose. After selecting it chose the General Tab, and select Connectors.

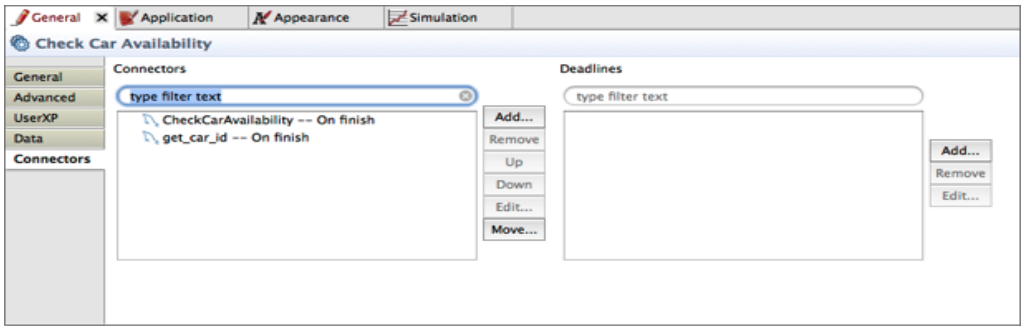


Figure 91: Create BPMN model on BOS - Add mysql connection

Select the Add button. A variety of connectors could be chosen; in this case the MySql connector is the right one.

Figure 92: Create BPMN model on BOS - New mysql connection

After that, a wizard is going to be shown so then the option details for the connection could be made.

A name will be asked and a possible description.

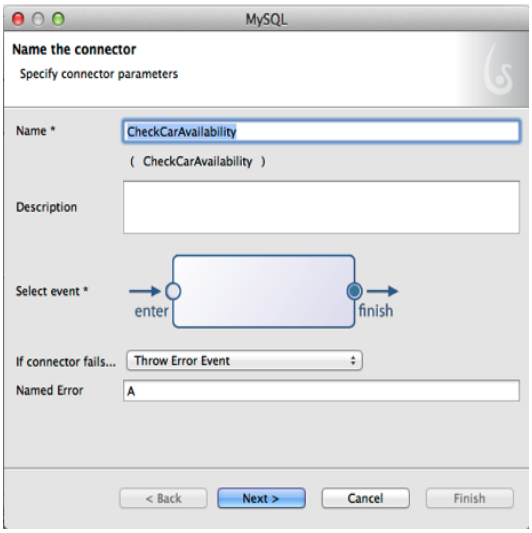


Figure 93: Create BPMN model on BOS - Add mysql name

A database and a server must be selected. A test configuration could be made in order to check database connectivity.

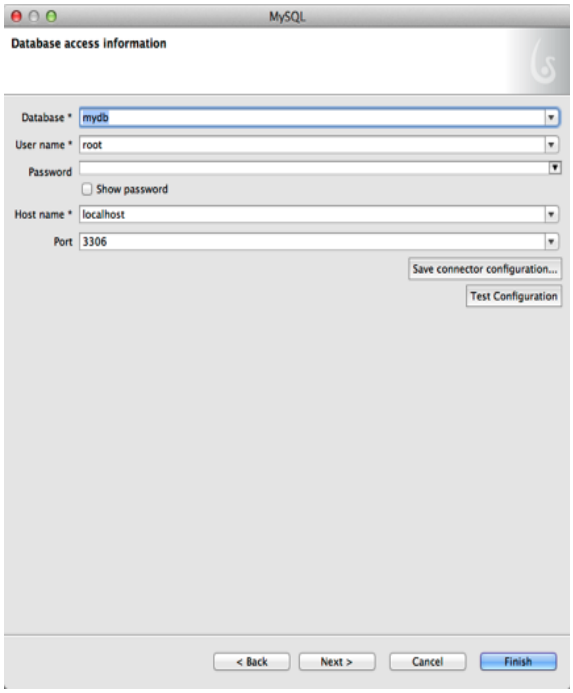


Figure 94: Create BPMN model on BOS - Mysql connection

An SQL function must be made in order to query the selected database. The function could be also tested and the results evaluated.

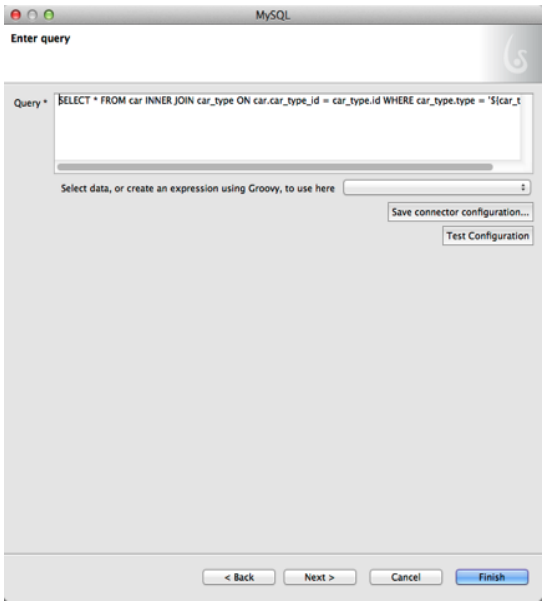


Figure 95: Create BPMN model on BOS - Adding mysql function

An output variable must be chosen so the output of the query is retain on the

process and used where is required.

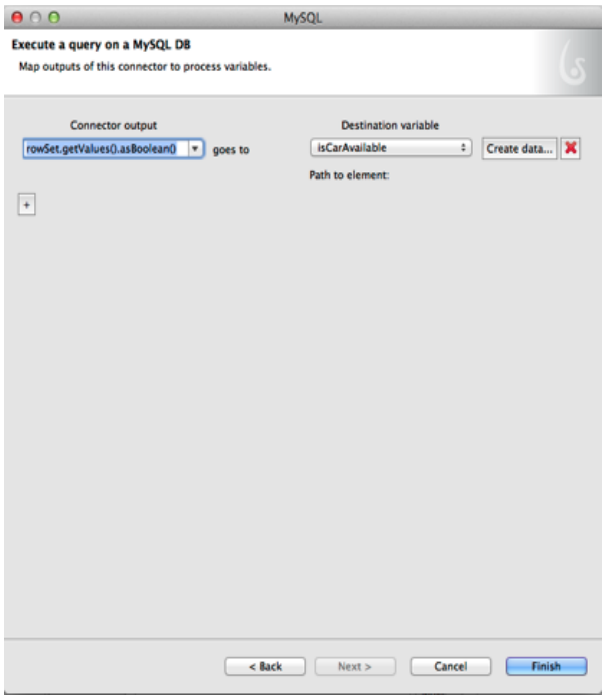


Figure 96: Create BPMN model on BOS - Connector output

B.2 Creating BPMN models on Activiti

There are two manners of creating a new model on the Activiti framework.

Is explained how to achieve the possibility of creating new models on both ways, but the real case example implementation and for the sake of this work, is made only on Eclipse Designer Plug-In.

Creating new model on Activiti Modeler Model Workspace

Having the Activiti 5.11 installed and ready, just need to have it running and the Processes tab selected.

Next, click the Model workspace.

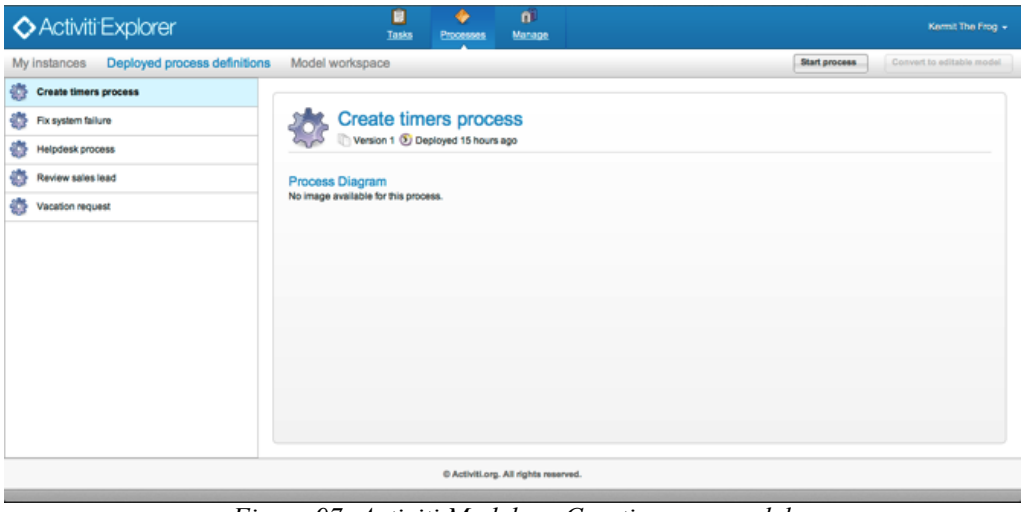


Figure 97: Activiti Modeler - Creating new model

Here is possible to find the available models on the system. There is the possibility of importing a model or even editing an already created model.

In this case the New Model is going to be selected.

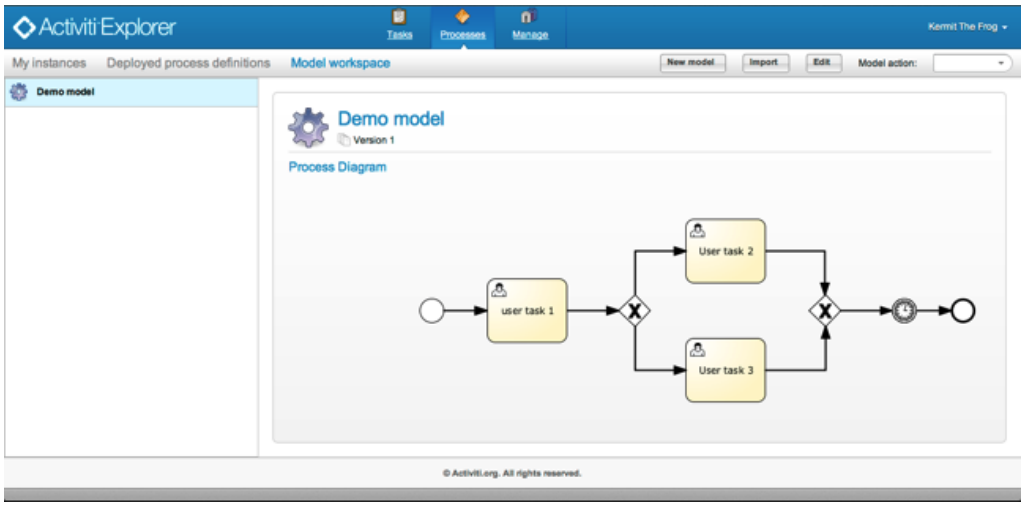


Figure 98: Activiti Modeler - Model workspace

Enter a name and an optional description and create the new model.

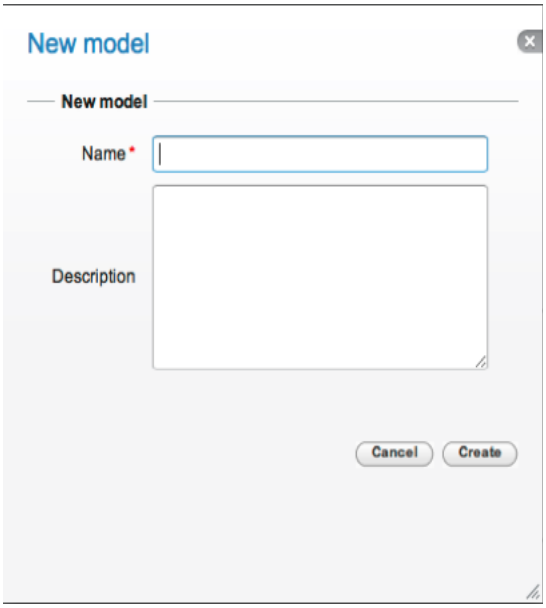


Figure 99: Activiti Modeler - New model

After that, the main dashboard of Activiti Modeler will be presented and with that, a new model could be designed.

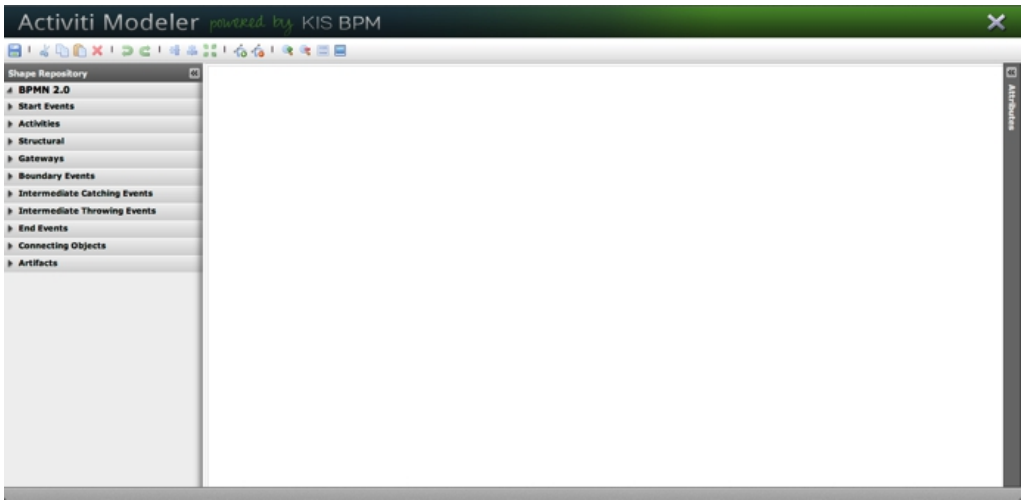


Figure 100: Activiti Modeler - Main dashboard

In the left one could find the main shape repository. On the right are the related attributes of the selected shape.

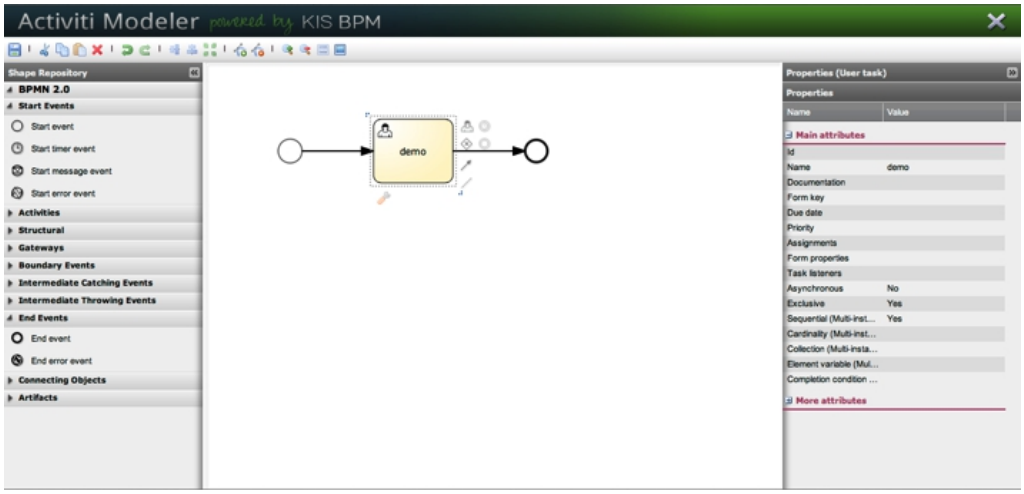


Figure 101: Activiti Modeler - Creating the model

Once finished one could simply close the Modeler on the “X” symbol on the right corner.

Returned to the Activiti Explorer one can chose to edit or even delete the created model just selecting the model, and the model action menu on the right.

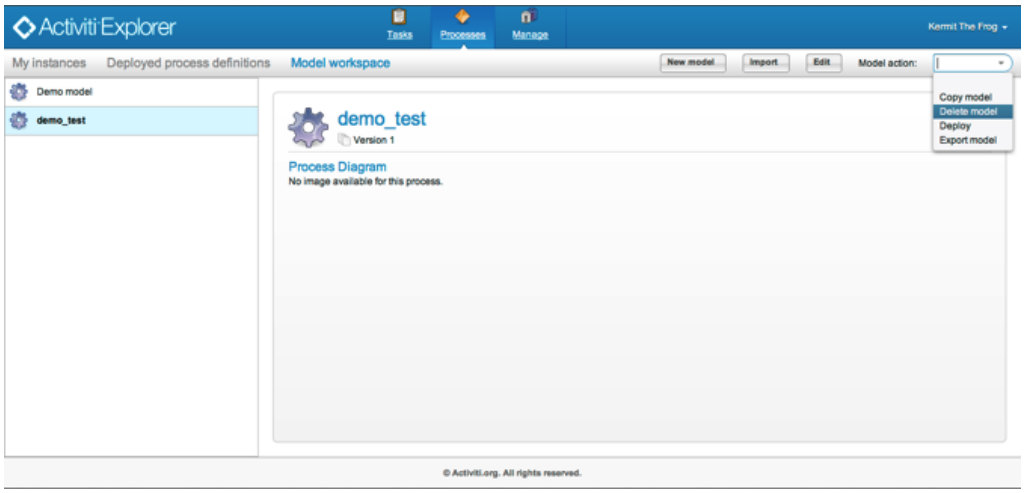


Figure 102: Activiti Modeler - Edit or delete a model

Creating new model on Activiti IDE Designer Plug-in

With the Ecplise IDE Designer installed and running just start a new project and select Activiti Projet.

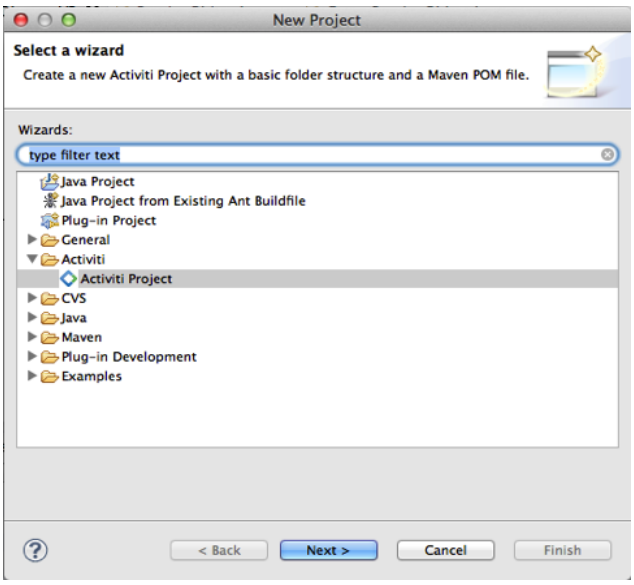


Figure 103: Activiti Designer plug-in - New project

Write a name and select a location to store it.

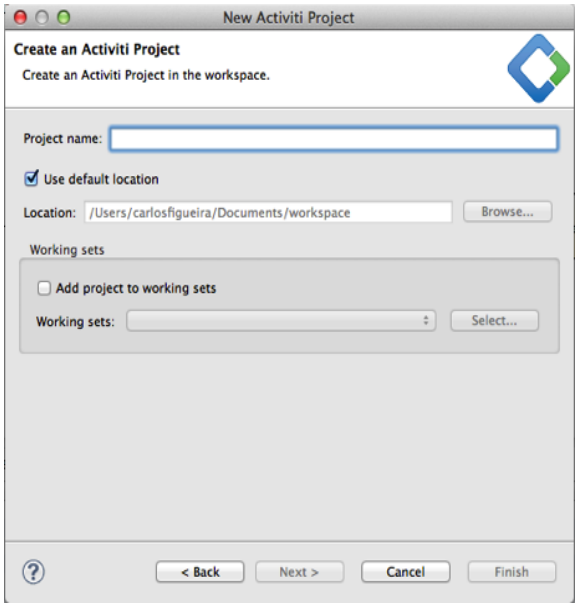


Figure 104: Activiti Designer plug-in - Creating activiti project

Add some optional references.

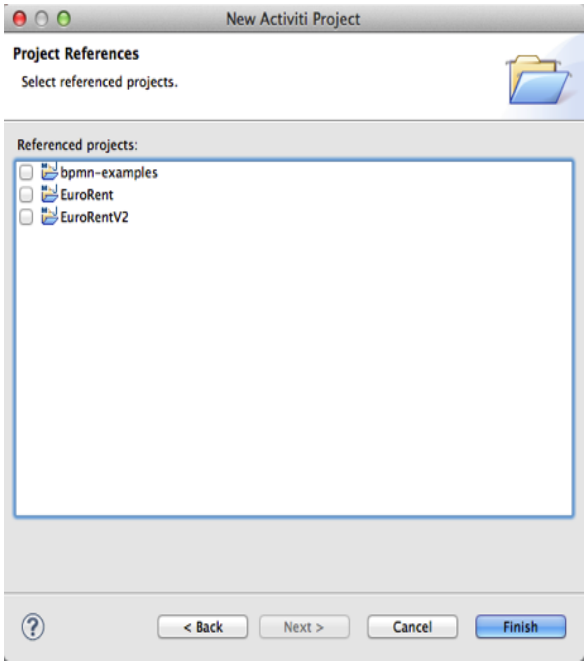


Figure 105: Activiti Designer plug-in - Adding optional references

After created the project just select it from the package explorer menu on the left.

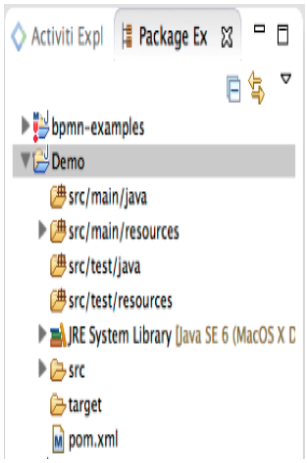


Figure 106: Activiti Designer plug-in - Project on package explorer

From the new file wizard select the Activiti Diagram so a BPMN diagram could be added to the project.

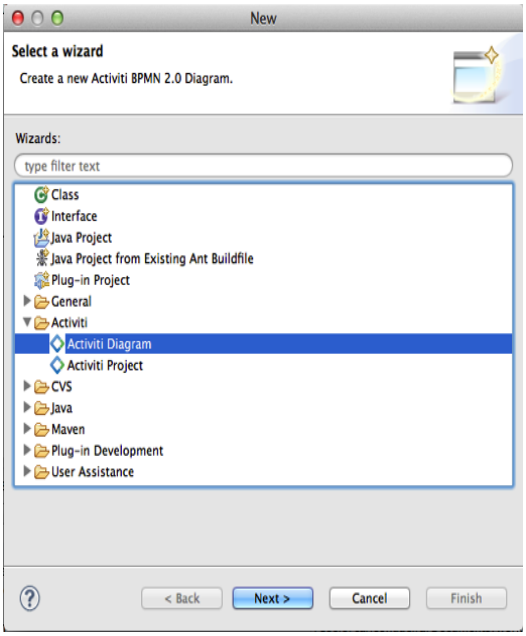


Figure 107: Activiti Designer plug-in - Add new BPMN diagram

Write the name and the location where the diagram would be added.

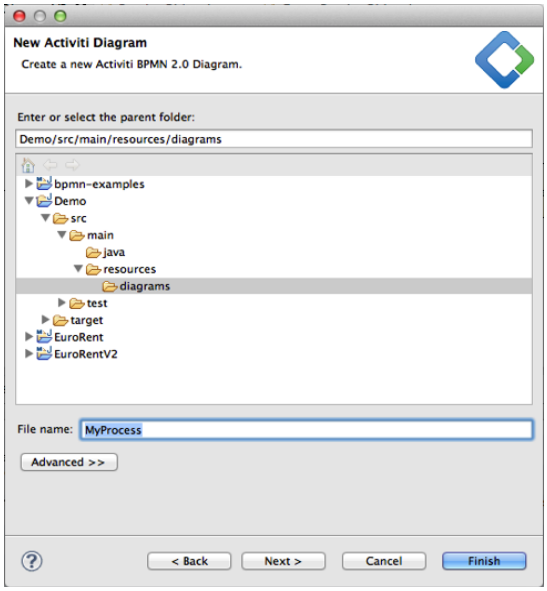


Figure 108: Activiti Designer plug-in - Save the diagram

Once the diagram is added to the project it is seen on the left menu.

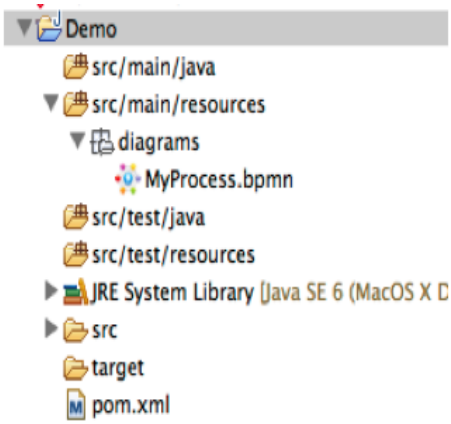


Figure 109: Activiti Designer plug-in - Add diagram to the project

With this, the diagram could be created using the BPMN shape pallet on the right.

Exporting (Deploying) the model to Activiti Explorer

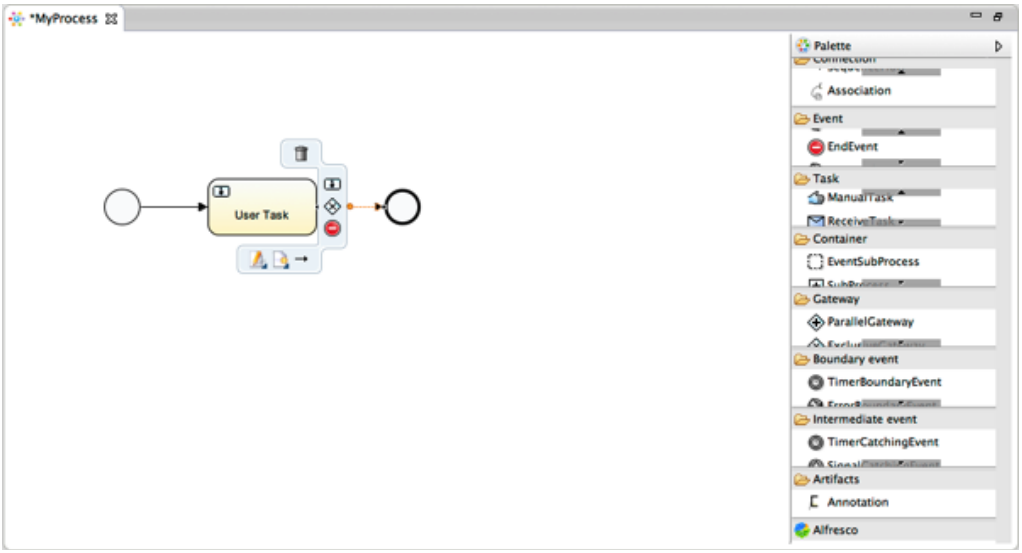


Figure 110: Activiti Designer plug-in - Add shapes to the diagram

Once the model is created, it can be exported and imported into the Activiti Explorer so then it could be executed.

For that, just right click on the project and select Create deployment Artifacts.

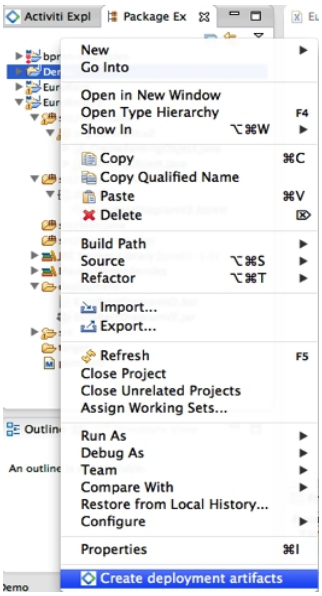


Figure 111: Activiti Designer plug-in - Export BPMN diagram

This step will create the necessary files to be imported on the Activiti Explorer. Just check the deployment directory on the project.

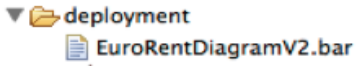


Figure 112: Activiti Designer plug-in - Deployed files

Importing deployed bpmn model to Activiti Explorer

After the creation of the .bar extension file, copy and paste it wherever desired so then it can be imported to Activiti Explorer.

On the Activiti Explorer main window click the Manage tab and select Deployments -> Upload New.

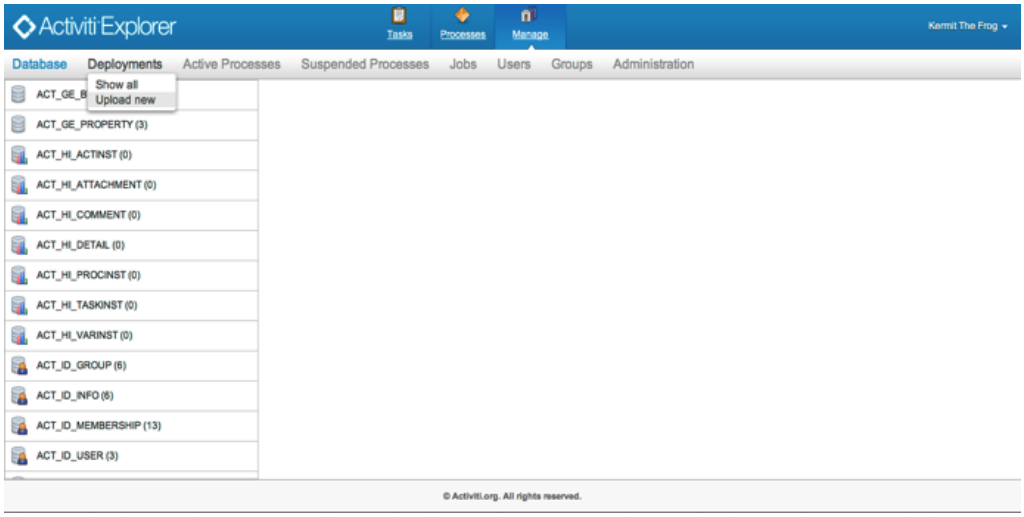


Figure 113: Importing to Activiti Explorer (Part 1)

Select the .bar file previously pasted on the selected directory to be uploaded.

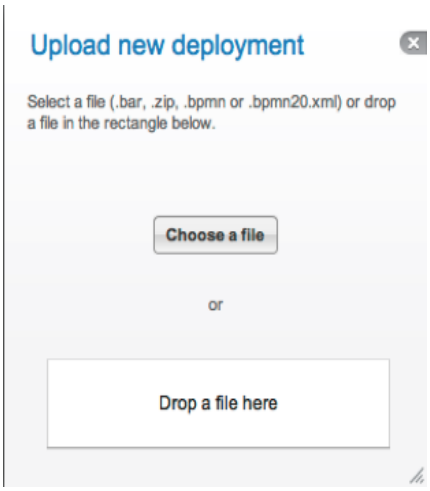


Figure 114: Importing to Activiti Explorer (Part 2)

If everything is fine with the model, it will be imported without any issues, however and if there is any problem it won't be imported at all, reporting its issue.

If well imported, just go to the next chapter in order to check how to execute the imported model.

Executing the model on Activiti Explorer

Once the model is imported without issues, it will appear on the deployments list.

Just select the process definition.

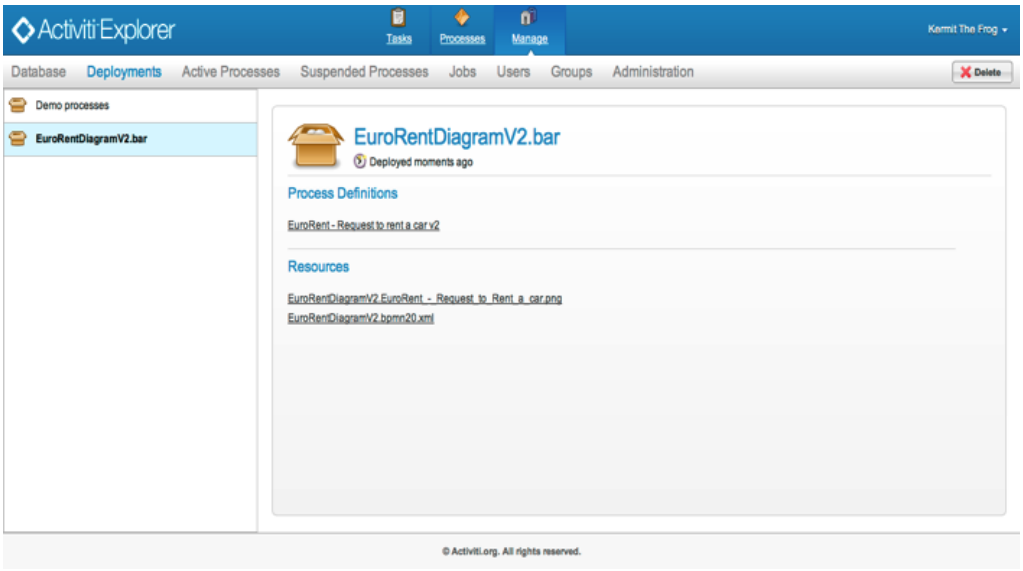


Figure 115: Execute model on Activiti - part 1

Selected the desired model, one can check the bpmn diagram and then select to start the process by clicking on the “Start Process” button on the right corner.

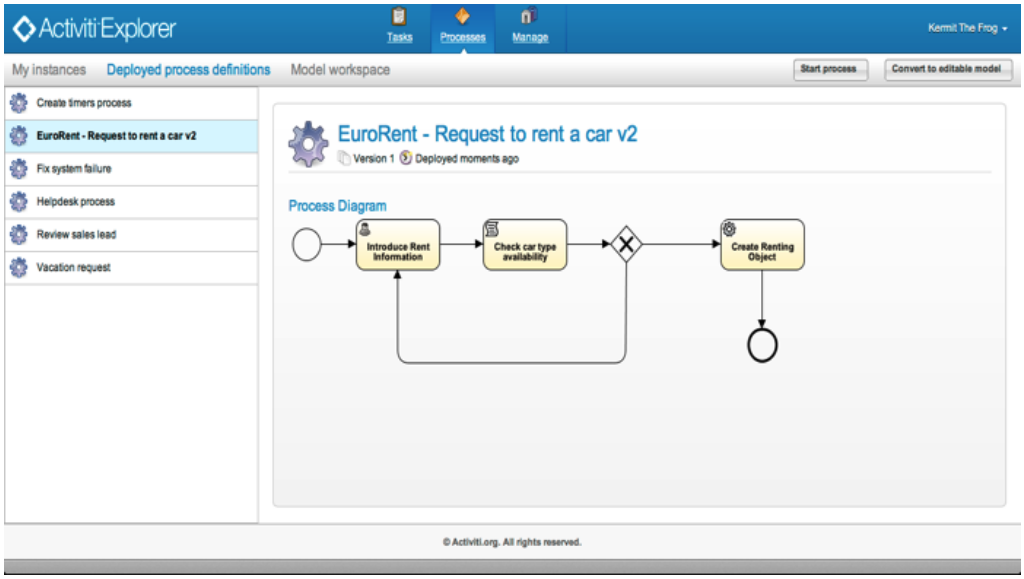


Figure 116: Execute model on Activiti - part 2

Started the process it goes directly to the first process on the diagram and to its related actor.

Since the one who started the process is the one who is directly related to the first process, it stays on the “involved” box.

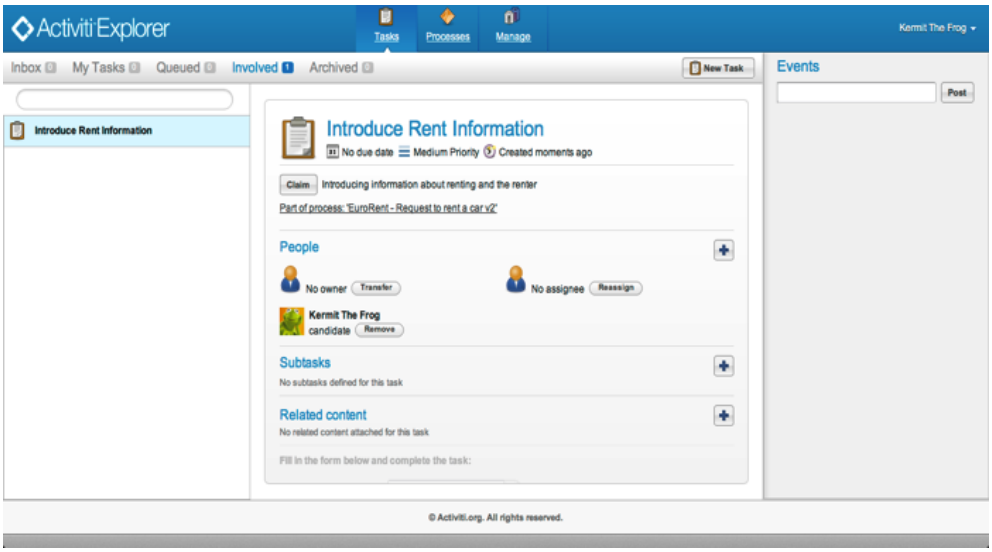


Figure 117: Execute model on Activiti - part 3

Once claimed the task of the first process, it goes to the inbox box.

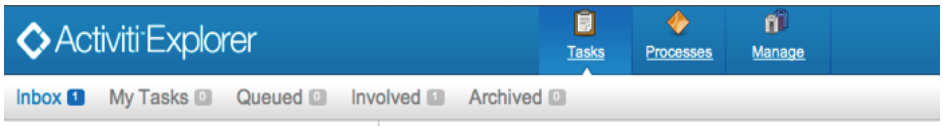


Figure 118: Execute model on Activiti - Claiming the tasks

With that, and in this case, the input data form is available to be fulfilled.

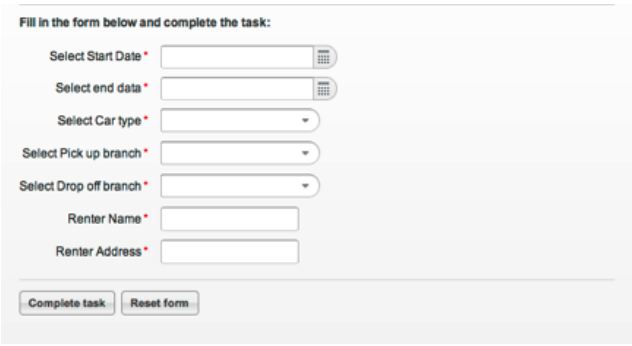


Figure 119: Execute model on Activiti - Input data form - part 1

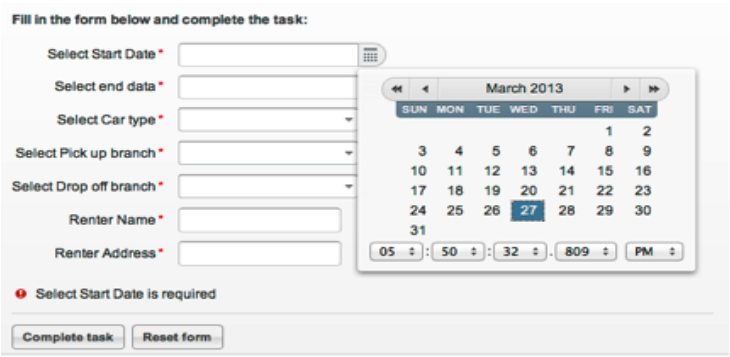


Figure 120: Execute model on Activiti - Input data form - part 2

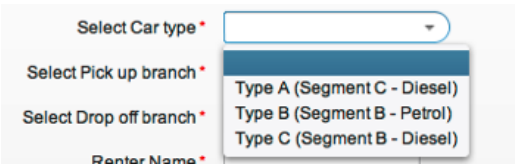


Figure 121: Execute model on Activiti - Input data form - part 3

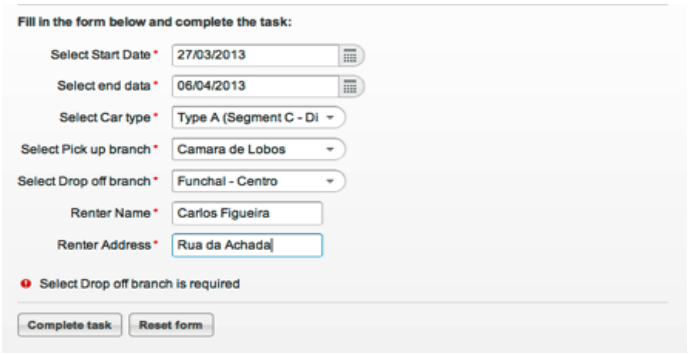


Figure 122: Execute model on Activiti - Input data form - part 4

Once fulfilled the input data form, and since the next processes are system processes, it gets completed. In this particular case only goes back if the desired car is not available for the desired dates.

Appendix C - EU-Rent use case

C.1 EU-Rent action rules new syntax

The EU-RENT actions rules were recreated using the new syntax proposed. Some of the already known rules had the needed of being reformulated and corrected so the rules can be created according the new syntax.

As mentioned before, each transaction have three main stages, the Order-Phase, the Execution-Phase and the Result-Phase. Each action rule is formulated regarding a particularly state of the transaction phase. The 1st rule shows all the actions taken when the *rental start of [rental] is requested*. When a rental start is requested by the renter a series of actions need to be made by the rental starter in order to the C-Act of the rental starter take place. In this particularly case the rental starter asks the renter for a set of documents and if the documents are right, the rental start must be promised, else the rental start is rejected, resulting in the abortion of the rental start.

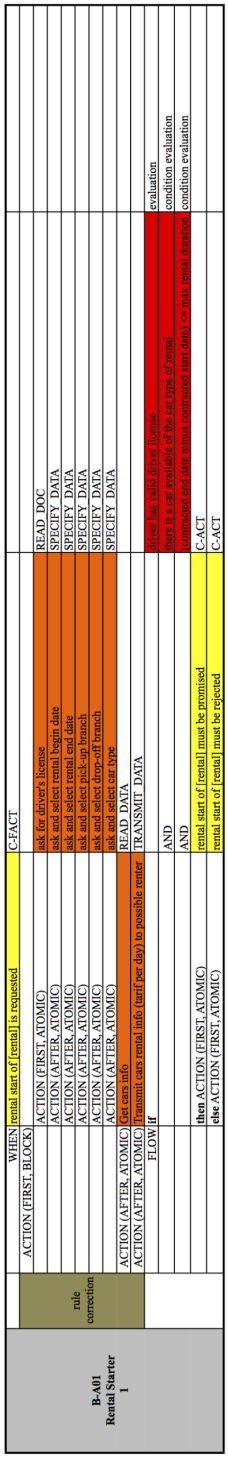


Figure 123: EU-Rent - Action rule 1 - New syntax

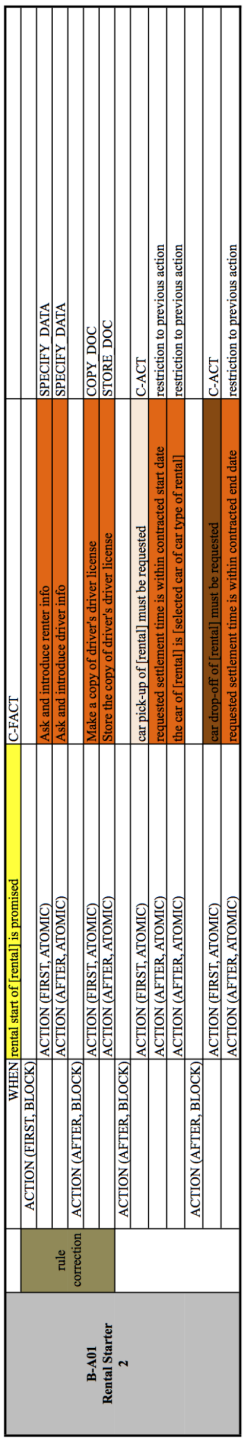


Figure 124: EU-Rent - Action rule 2 - New syntax

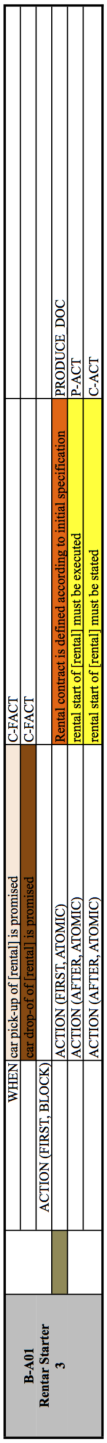


Figure 125: EU-Rent - Action rule 3 - New syntax

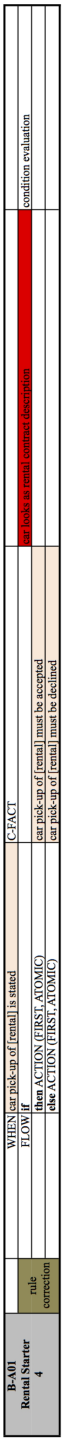


Figure 126: EU-Rent - Action rule 4 - New syntax

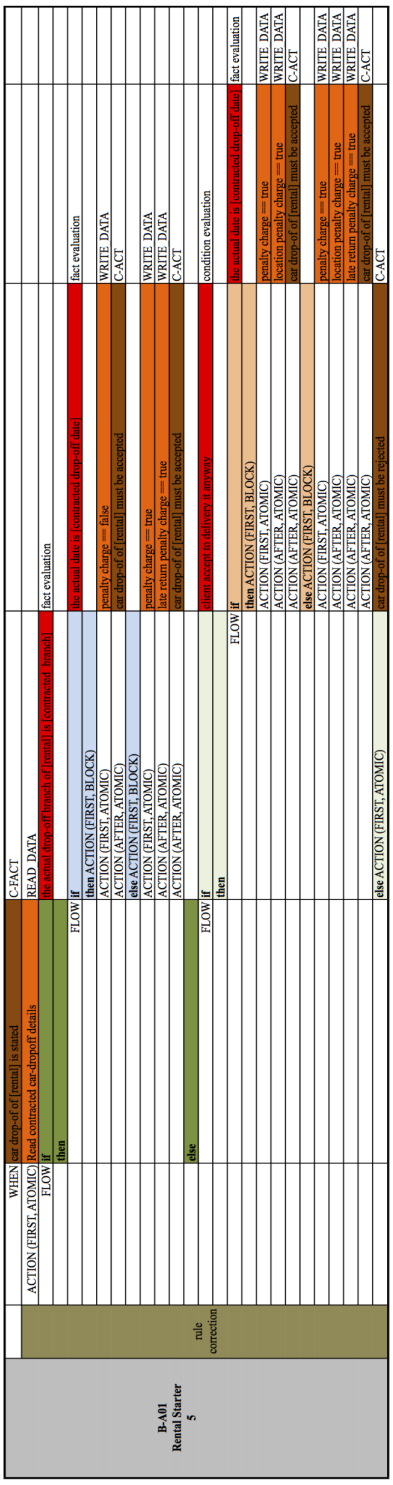


Figure 127: EU-Rent - Action rule 5 - New syntax

Figure 128: EU-Rent - Action rule 6 - New syntax

Figure 129: EU-Rent - Action rule 7 - New syntax

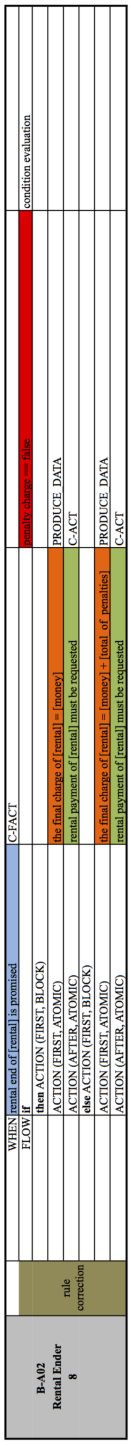


Figure 130: EU-Rent - Action rule 8 - New syntax

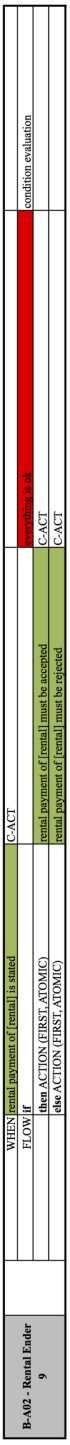


Figure 131: EU-Rent - Action rule 9 - New syntax

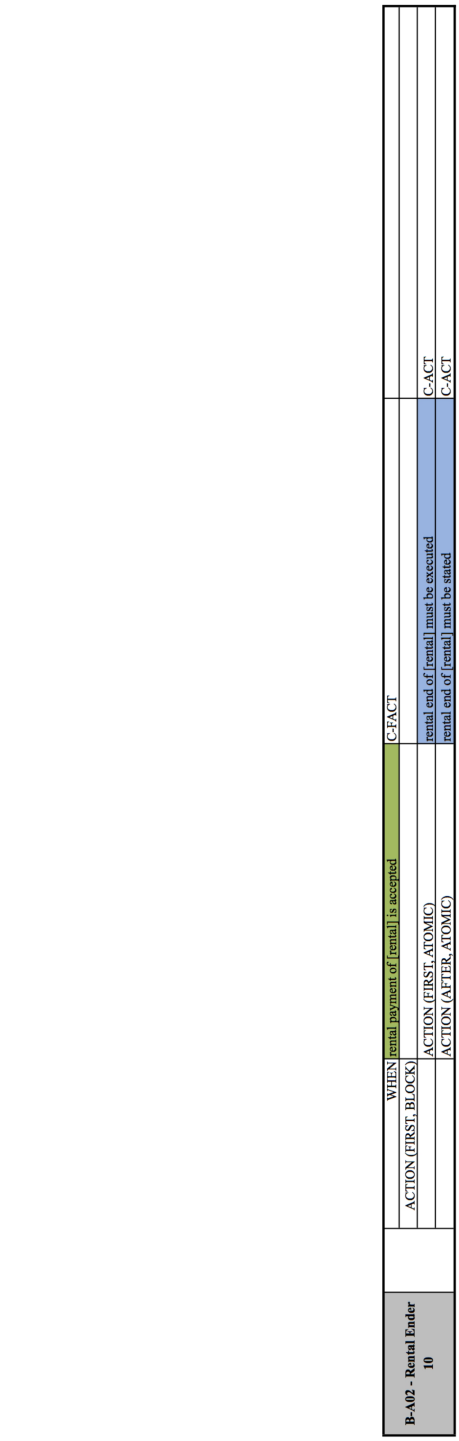


Figure 132: EU-Rent - Action rule 10 - New syntax

Figure 133: EU-Rent - Action rule 11 - New syntax

This new set of action rules made according the new syntax shows how all the ontological, datalogical and infological acts are part of any actor even without interfering with the implementation part of the process.

With all this information on the action rules, is possible to manually create on BOS a compliant and executable BPMN process based on the DEMO models of the EU-Rent, totally independent from its final implementation. During the implementation, the one who implements it, is the one who needs to decide how the datalogical and infological acts should be made, depending of each particularly implementation.

C.2 DEMO models into UEAOM based BPMN workflow

With the new action rule syntax, the transformation and generation of a compliant BPMN model with the DEMO aspect models is possible to be made. For each transaction there is a process, composed by two lanes, each lane represents the actor involved on the transaction that according to DEMO is the addressee and the addresser. All this construction was manual and totally based on the new action rule syntax and the suggested guidelines.

The following figures there are depicted all the processes constructed on Bonita Software for the EU-RENT case.

Figure 134 represents the process regarding the transaction T01 – rental start. As said before, there are two lanes, each one representing the involved actors in the transaction, in this case the external B-CA01 – Renter and the internal B-A01 - Rental Starter. The B-CA01 external actor is the responsible for enacting the transaction with the C-ACT T01 - request. Since there are always a C-FACT result of a C-ACT, there is the C-FACT T01 is requested. There are the throw and catch events representing those happenings on the diagram, as there will be in all the processes, as seen on the guidelines for the BPMN construction.

Following exactly what was depicted on the action rules, according to the new syntax, the BPMN processes are simple to construct and still maintaining its implementation independence.

As seen in the Action Rule depicted on Figure 123 regarding the Transaction T01 - rental start - there are several actions, flows, conditions and C-ACT's. When T01 is requested there is a Block of actions, meaning that a group of actions is made sequentially without any interruptions. In the BPMN diagram that block of actions is represented by the Read_doc/Specify_data event. Then there is two atomic actions, meaning that they must happen individually. Those individual actions are represented by the Read_data and the Transmit_data events. Then, proceeds with a Flow (IF) verifying a condition. The conditions are represented as events. If the result of the vent is true for all the conditions events the flow proceeds for the C-ACT promised, else the C-ACT is rejected. All the C-ACT's are represented as Throw and Catch events.

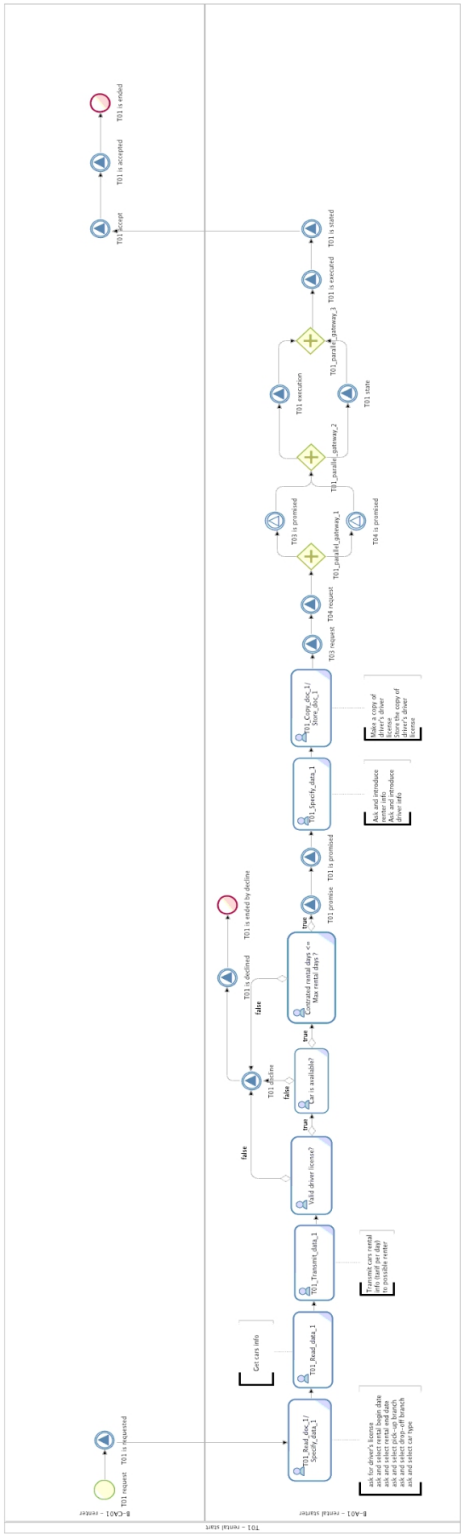


Figure 134: BPMN model - T01 rental start

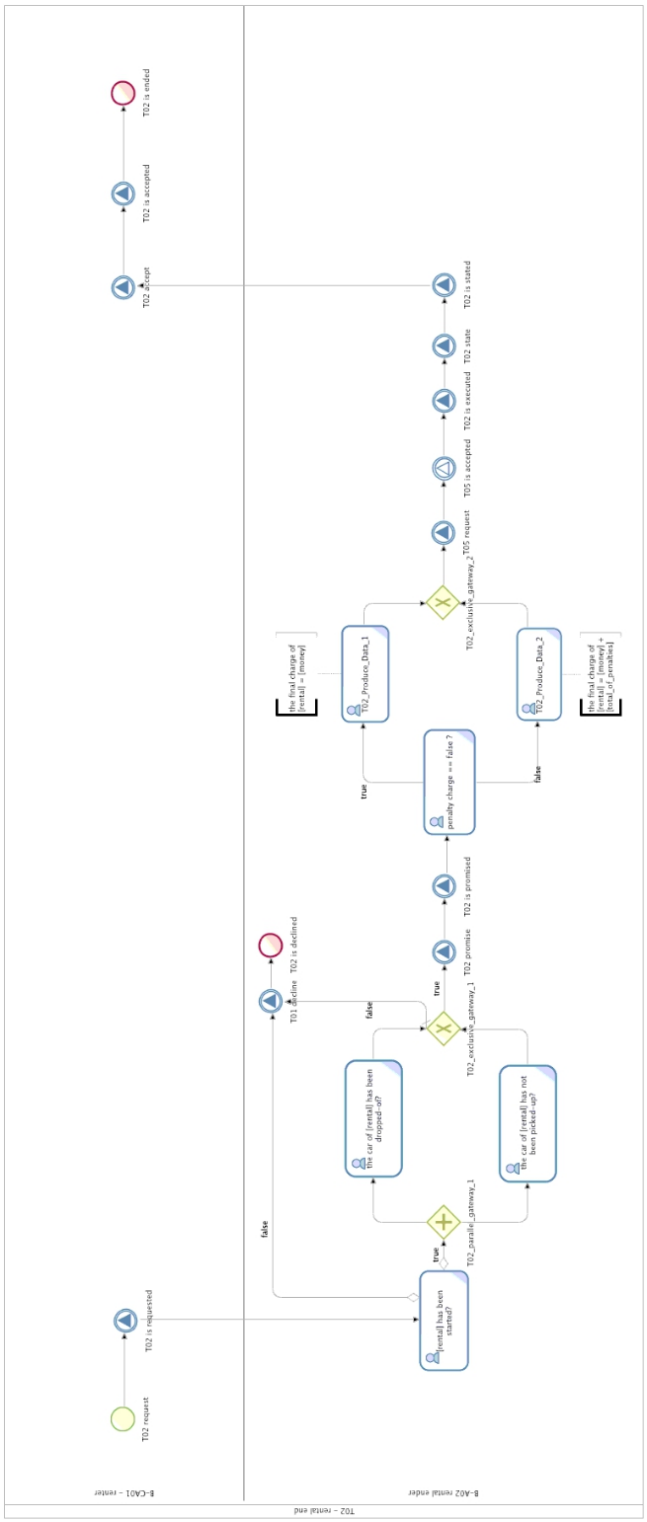


Figure 135: BPMN model - T02 rental end

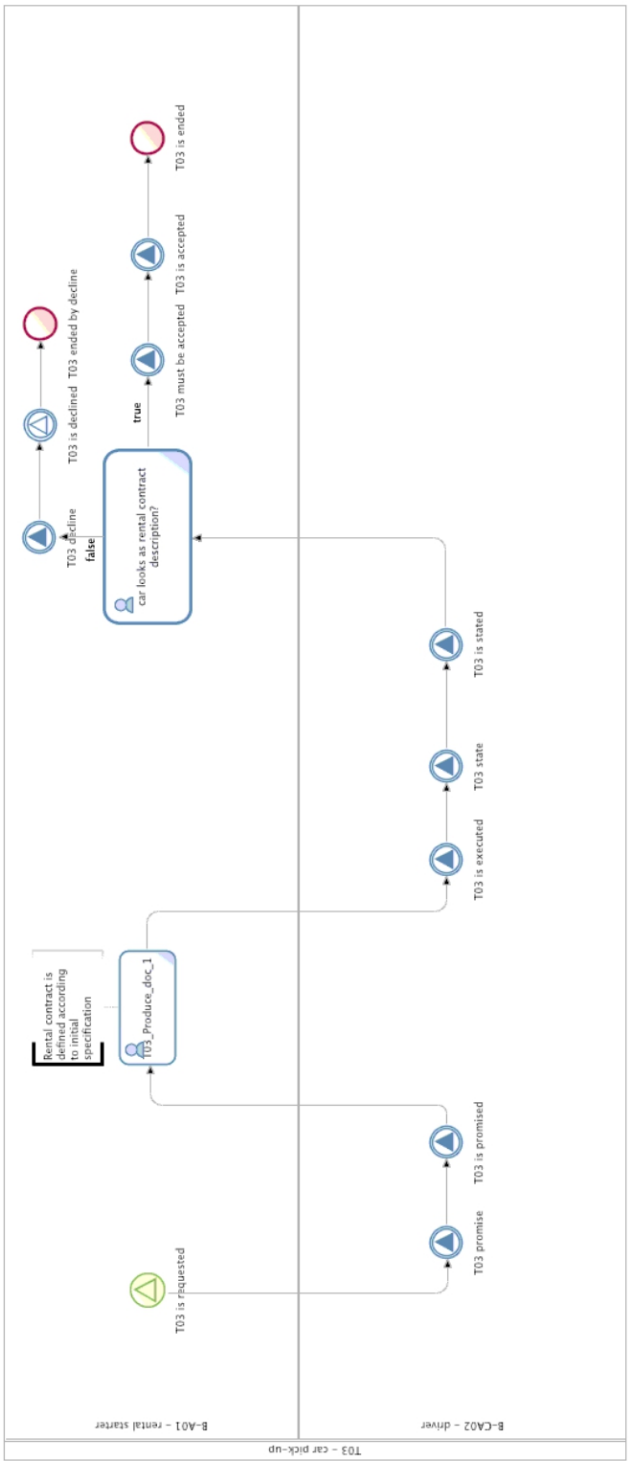


Figure 136: BPMN model - T03 car pickup

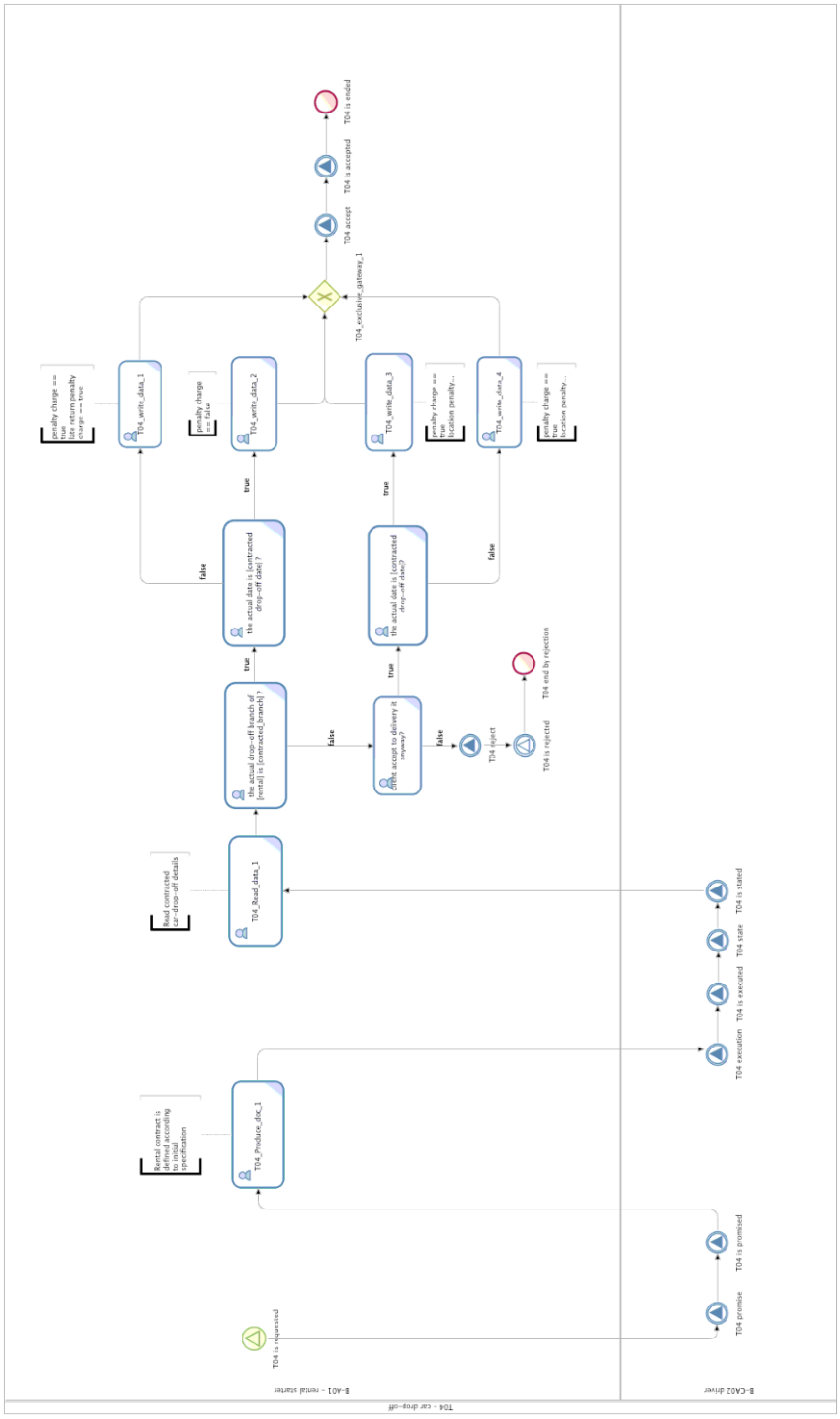


Figure 137: BPMN model - T04 car drop off

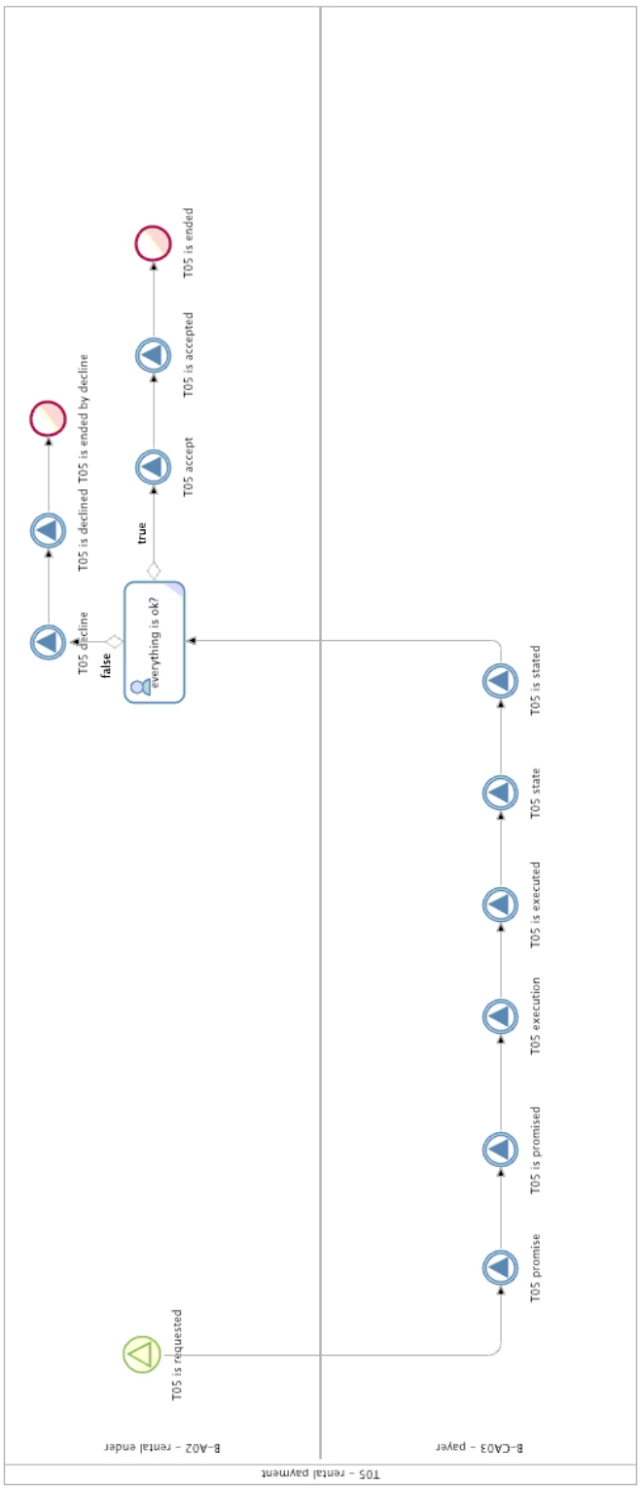


Figure 138: BPMN model - T05 rental payment

C.3 EU-Rent action rules UEAOM instantiation

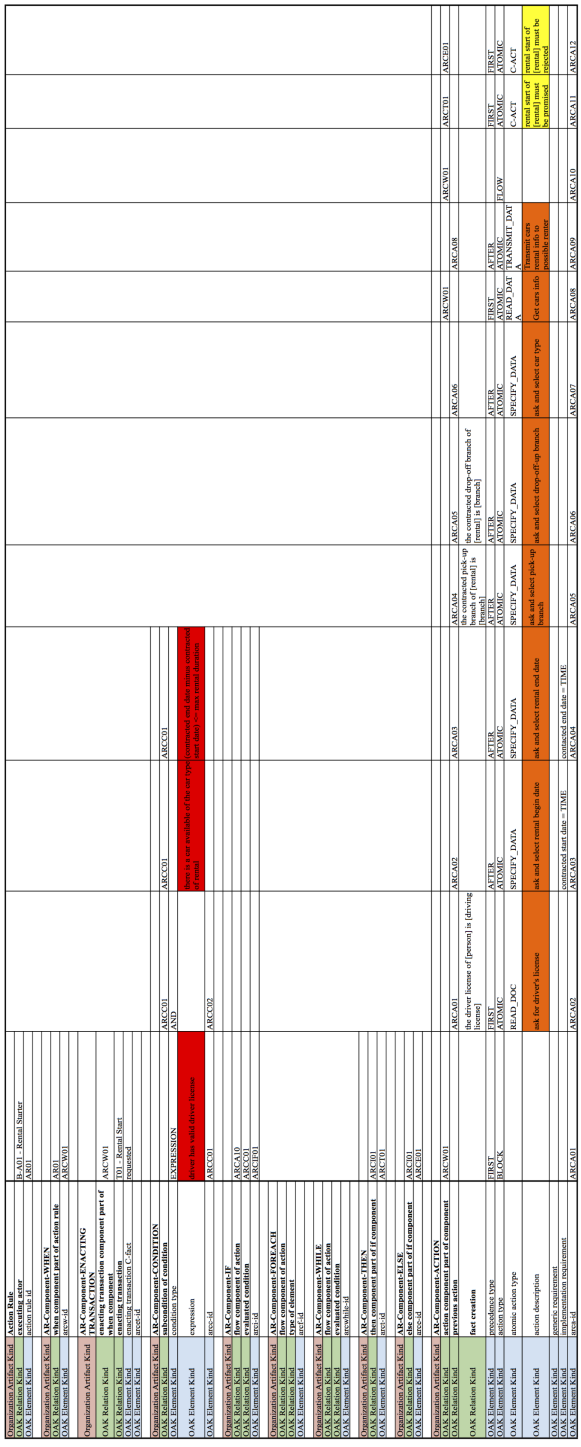


Figure 139: EU-Rent - Action rule 1 - UEAOM instantiation

Organization Artifact Kind	**Action Rule**	B-A01 - Rental Starter																							
OAK Relation Kind	**executing actor**	AR02																							
OAK Element Kind	action rule id																								
Organization Artifact Kind	**AR-Component-WHEN**																								
OAK Relation Kind	**when component part of action rule**	AR02																							
OAK Element Kind	arcw-id	ARCW01																							
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**																								
OAK Relation Kind	**enacting transction component part of when component**	ARCW01																							
OAK Relation Kind	**enacting Transaction**	T01 - Rental Start																							
OAK Element Kind	enacting transaction C-fact	promised																							
OAK Element Kind	arcet-id	ARCET01																							
Organization Artifact Kind	**AR-Component-CONDITION**																								
OAK Relation Kind	**subcondition of condition**																								
OAK Element Kind	condition type																								
OAK Element Kind	expression																								
OAK Element Kind	arcc-id																								
Organization Artifact Kind	**AR-Component-IF**																								
OAK Relation Kind	**if component part of action**																								
OAK Relation Kind	**evaluated condition**																								
OAK Element Kind	arci-id																								
Organization Artifact Kind	**AR-Component-FOREACH**																								
OAK Relation Kind	**foreach component of action**																								
OAK Relation Kind	**type of element**																								
OAK Element Kind	arcf-id																								
Organization Artifact Kind	**AR-Component-WHILE**																								
OAK Relation Kind	**flow component of action**																								
OAK Relation Kind	**evaluated condition**																								
OAK Element Kind	arcwhile-id																								
Organization Artifact Kind	**AR-Component-THEN**																								
OAK Relation Kind	**then component part of if component**																								
OAK Element Kind	arci-id																								
Organization Artifact Kind	**AR-Component-ELSE**																								
OAK Relation Kind	**else component part of if component**																								
OAK Element Kind	arce-id																								
Organization Artifact Kind	**AR-Component-ACTION**	ARCW01						ARCW01				ARCW01										ARCW01			
OAK Relation Kind	**action component part of component**		ARCA01		ARCA02			ARCA03				ARCA04				ARCA07		ARCA08		ARCA09		ARCA07		ARCA11	ARCA12
OAK Relation Kind	**precious action**		the renter of [rental] is [person]		the owner of [rental] is [person]			the owner of [rental] is [person]																	
OAK Element Kind	precedence type	FIRST	the renter of [rental] is [person]		AFTER			AFTER				FIRST				AFTER		FIRST		AFTER		AFTER		FIRST	AFTER
OAK Element Kind	action type	BLOCK	ATOMIC		ATOMIC			BLOCK				ATOMIC				ATOMIC		ATOMIC		ATOMIC		BLOCK		ATOMIC	ATOMIC
OAK Element Kind	atomic action type		SPECIFY_DATA		SPECIFY_DATA							COPY_DOC				STORE_DOC		C-ACT		C-ACT				C-ACT	
OAK Element Kind	action description		Ask and introduce renter info		Ask and introduce driver info							Make a copy of driver's driver license				Make a copy of driver's driver license		car pick-up of [rental] must be requested						car drop-off of [rental] must be requested	
OAK Element Kind	generic requirement																								
OAK Element Kind	implementation requirement																	requested settlement time is within contracted rental data		the car of [rental] is located at [car type of rental]				requested settlement time is within contracted rental data	
OAK Element Kind	arca-id	ARCA01		ARCA02		ARCA03			ARCA04				ARCA05		ARCA06		ARCA07		ARCA08		ARCA10		ARCA11		ARCA12	ARCA13

Figure 140: EU-Rent - Action rule 2 - UEAOM instantiation

Organization Artifact Kind	**Action Rule**					
OAK Relation Kind	**executing actor**	B-A01 - Rental Starter				
OAK Element Kind	action rule id	AR03				
Organization Artifact Kind	**AR-Component-WHEN**					
OAK Relation Kind	**when component part of action rule**	AR03				
OAK Element Kind	arcw-id	ARCW01				
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**					
OAK Relation Kind	**enacting transaction component part of when component**	ARCW01	ARCW01			
OAK Relation Kind	**enacting transaction**	T03 - Car pick-up	T04 - car drop-of			
OAK Element Kind	enacting transaction C-fact	promised	promised			
OAK Element Kind	arcet-id	ARCET01	ARCET02			
Organization Artifact Kind	**AR-Component-CONDITION**					
OAK Relation Kind	**subcondition of condition**					
OAK Element Kind	condition type					
OAK Element Kind	expression					
OAK Element Kind	arcc-id					
Organization Artifact Kind	**AR-Component-IF**					
OAK Relation Kind	**flow component of action**					
OAK Relation Kind	**evaluated condition**					
OAK Element Kind	arci-id					
Organization Artifact Kind	**AR-Component-FOREACH**					
OAK Relation Kind	**flow component of action**					
OAK Relation Kind	**type of element**					
OAK Element Kind	arcf-id					
Organization Artifact Kind	**AR-Component-WHILE**					
OAK Relation Kind	**flow component of action**					
OAK Relation Kind	**evaluated condition**					
OAK Element Kind	arcwhile-id					
Organization Artifact Kind	**AR-Component-THEN**					
OAK Relation Kind	**then component part of if component**					
OAK Element Kind	arct-id					
Organization Artifact Kind	**AR-Component-ELSE**					
OAK Relation Kind	**else component part of if component**					
OAK Element Kind	arce-id					
Organization Artifact Kind	**AR-Component-ACTION**					
OAK Relation Kind	**action component part of component**	ARCW01	ARCW01	ARC02	ARCA03	
OAK Relation Kind	**previous action**					
OAK Relation Kind	**fact creation**					
OAK Element Kind	precedence type	FIRST	FIRST	AFTER	AFTER	
OAK Element Kind	action type	BLOCK	ATOMIC	ATOMIC	ATOMIC	
OAK Element Kind	atomic action type		PRODUCE_DOC	P-ACT	C-ACT	
OAK Element Kind	action description		Rental contract is defined according to initial specification	rental start of [rental] must be executed	rental start of [rental] must be stated	
OAK Element Kind	generic requirement					
OAK Element Kind	implementation requirement					
OAK Element Kind	arca-id	ARCA01	ARC02	ARCA03	ARCA04	

Figure 141: EU-Rent - Action rule 3 - UEAOM instantiation

Organization Artifact Kind	**Action Rule**			
OAK Relation Kind	**executing actor**	B-A01 - Rental Starter		
OAK Element Kind	action rule id	AR04		
Organization Artifact Kind	**AR-Component-WHEN**			
OAK Relation Kind	**when component part of action rule**	AR04		
OAK Element Kind	arcw-id	ARCW01		
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**			
OAK Relation Kind	**enacting transaction component part of when component**	ARCW01		
OAK Relation Kind	**enacting transaction**	T03 - Car pick-up		
OAK Element Kind	enacting transaction C-fact	stated		
OAK Element Kind	arcet-id	ARCET01		
Organization Artifact Kind	**AR-Component-CONDITION**			
OAK Relation Kind	**subcondition of condition**			
OAK Element Kind	condition type	EXPRESSION		
OAK Element Kind	expression	car looks as rental contract description		
OAK Element Kind	arcc-id	ARCC01		
Organization Artifact Kind	**AR-Component-IF**			
OAK Relation Kind	**flow component of action**	ARCA01		
OAK Relation Kind	**evaluated condition**	ARCC01		
OAK Element Kind	arci-id	ARCIF01		
Organization Artifact Kind	**AR-Component-FOREACH**			
OAK Relation Kind	**flow component of action**			
OAK Relation Kind	**type of element**			
OAK Element Kind	arcf-id			
Organization Artifact Kind	**AR-Component-WHILE**			
OAK Relation Kind	**flow component of action**			
OAK Relation Kind	**evaluated condition**			
OAK Element Kind	arcwhile-id			
Organization Artifact Kind	**AR-Component-THEN**			
OAK Relation Kind	**then component part of if component**	ARCIF01		
OAK Element Kind	arct-id	ARCT01		
Organization Artifact Kind	**AR-Component-ELSE**			
OAK Relation Kind	**else component part of if component**	ARCIF01		
OAK Element Kind	arce-id	ARCE01		
Organization Artifact Kind	**AR-Component-ACTION**			
OAK Relation Kind	**action component part of component**	ARCW01	ARCT01	ARCE01
OAK Relation Kind	**previous action**			
OAK Relation Kind	**fact creation**			
OAK Element Kind	precedence type		FIRST	FIRST
OAK Element Kind	action type	FLOW	ATOMIC	ATOMIC
OAK Element Kind	atomic action type		C-ACT	C-ACT
OAK Element Kind	action description		car pick-up of [rental] must be accepted	car pick-up of [rental] must be declined
OAK Element Kind	generic requirement			
OAK Element Kind	implementation requirement			
OAK Element Kind	arca-id	ARCA01	ARCA02	ARCA03

Figure 142: EU-Rent - Action rule 4 - UEAOM instantiation

Figure 143: EU-Rent - Action rule 5 - UEAOM instantiation

Organization Artifact Kind	**Action Rule**	
OAK Relation Kind	**executing actor**	B-A01 - Rental Starter
OAK Element Kind	action rule id	AR06
Organization Artifact Kind	**AR-Component-WHEN**	
OAK Relation Kind	**when component part of action rule**	AR06
OAK Element Kind	arcw-id	ARCW01
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**	
OAK Relation Kind	**enacting transaction component part of when component**	ARCW01
OAK Relation Kind	**enacting transaction**	T01 - Rental Start
OAK Element Kind	enacting transaction C-fact	stated
OAK Element Kind	arcet-id	ARCET01
Organization Artifact Kind	**AR-Component-CONDITION**	
OAK Relation Kind	**subcondition of condition**	
OAK Element Kind	condition type	
OAK Element Kind	expression	
OAK Element Kind	arcc-id	
Organization Artifact Kind	**AR-Component-IF**	
OAK Relation Kind	**flow component of action**	
OAK Relation Kind	**evaluated condition**	
OAK Element Kind	arci-id	
Organization Artifact Kind	**AR-Component-FOREACH**	
OAK Relation Kind	**flow component of action**	
OAK Relation Kind	**type of element**	
OAK Element Kind	arcf-id	
Organization Artifact Kind	**AR-Component-WHILE**	
OAK Relation Kind	**flow component of action**	
OAK Relation Kind	**evaluated condition**	
OAK Element Kind	arcwhile-id	
Organization Artifact Kind	**AR-Component-THEN**	
OAK Relation Kind	**then component part of if component**	
OAK Element Kind	arct-id	
Organization Artifact Kind	**AR-Component-ELSE**	
OAK Relation Kind	**else component part of if component**	
OAK Element Kind	arce-id	
Organization Artifact Kind	**AR-Component-ACTION**	
OAK Relation Kind	**action component part of component**	ARCW01
OAK Relation Kind	**previous action**	
OAK Relation Kind	**fact creation**	
OAK Element Kind	precedence type	FIRST
OAK Element Kind	action type	ATOMIC
OAK Element Kind	atomic action type	C-Act
OAK Element Kind	action description	rental start of [rental] must be accepted
OAK Element Kind	generic requirement	
OAK Element Kind	implementation requirement	
OAK Element Kind	arca-id	ARCA01

Figure 144: EU-Rent - Action rule 6 - UEAOM instantiation

Organization Artifact Kind	**Action Rule**	B-A02 - Rental Ender							
OAK Relation Kind	**executing actor**	AR07							
OAK Element Kind	action rule id								
Organization Artifact Kind	**AR-Component-WHEN**								
OAK Relation Kind	**when component part of action rule**	AR07							
OAK Element Kind	arcw-id	ARCW01							
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**								
OAK Relation Kind	**enacting transaction component part of when component**	ARCW01							
OAK Relation Kind	**enacting transaction**	T02 - Rental End							
OAK Element Kind	enacting transaction C-fact	requested							
OAK Element Kind	arcet-id	ARCET01							
Organization Artifact Kind	**AR-Component-CONDITION**								
OAK Relation Kind	**subcondition of condition**		ARCC01		ARCC01		ARCC04	ARCC04	
OAK Element Kind	condition type	AND	Expression		Expression	OR	NOT	Expression	
OAK Element Kind	expression		[rental] has been started		the car of [rental] has been dropped-of			the car of [rental] has been picked-up	
OAK Element Kind	arcc-id	ARCC01	ARCC02		ARCC03	ARCC04	ARCC05	ARCC06	
Organization Artifact Kind	**AR-Component-IF**								
OAK Relation Kind	**flow component of action**	ARCA01							
OAK Relation Kind	**evaluated condition**	ARCC01							
OAK Element Kind	arci-id	ARCIF01							
Organization Artifact Kind	**AR-Component-FOREACH**								
OAK Relation Kind	**flow component of action**								
OAK Relation Kind	**type of element**								
OAK Element Kind	arcf-id								
Organization Artifact Kind	**AR-Component-WHILE**								
OAK Relation Kind	**flow component of action**								
OAK Relation Kind	**evaluated condition**								
OAK Element Kind	arcwhile-id								
Organization Artifact Kind	**AR-Component-THEN**								
OAK Relation Kind	**then component part of if component**	ARCIF01							
OAK Element Kind	arct-id	ARCT01							
Organization Artifact Kind	**AR-Component-ELSE**								
OAK Relation Kind	**else component part of if component**	ARCIF01							
OAK Element Kind	arce-id	ARCE01							
Organization Artifact Kind	**AR-Component-ACTION**								
OAK Relation Kind	**action component part of component**	ARCW01	ARCT01		ARCE01				
OAK Relation Kind	**previous action**								
OAK Relation Kind	**fact creation**								
OAK Element Kind	precedence type		FIRST		FIRST				
OAK Element Kind	action type	FLOW	ATOMIC		ATOMIC				
OAK Element Kind	atomic action type		C-Act		C-Act				
OAK Element Kind	action description		rental end of [rental] must be promised		rental start of [rental] must be rejected				
OAK Element Kind	generic requirement								
OAK Element Kind	implementation requirement								
OAK Element Kind	arca-id	ARCA01	ARCA02		ARCA03				

Figure 145: EU-Rent - Action rule 7 - UEAOM instantiation

Organization Artifact Kind	**Action Rule**										
OAK Relation Kind	**executing actor**	B-A02 - Rental Ender									
OAK Element Kind	action rule id	AR08									
Organization Artifact Kind	**AR-Component-WHEN**										
OAK Relation Kind	**when component part of action rule**	AR08									
OAK Element Kind	arcw-id	ARCW01									
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**										
OAK Relation Kind	**enacting transaction component part of when component**	ARCW01									
OAK Relation Kind	**enacting transaction**	T02 - Rental End									
OAK Element Kind	enacting transaction C-fact	promised									
OAK Element Kind	arcet-id	ARCET01									
Organization Artifact Kind	**AR-Component-CONDITION**										
OAK Relation Kind	**subcondition of condition**										
OAK Element Kind	condition type	Expression									
OAK Element Kind	expression	penalty charge == false									
OAK Element Kind	arcc-id	ARCC01									
Organization Artifact Kind	**AR-Component-IF**										
OAK Relation Kind	**flow component of action**	ARCA01									
OAK Relation Kind	**evaluated condition**	ARCC01									
OAK Element Kind	arci-id	ARCIF01									
Organization Artifact Kind	**AR-Component-FOREACH**										
OAK Relation Kind	**flow component of action**										
OAK Relation Kind	**type of element**										
OAK Element Kind	arcf-id										
Organization Artifact Kind	**AR-Component-WHILE**										
OAK Relation Kind	**flow component of action**										
OAK Relation Kind	**evaluated condition**										
OAK Element Kind	arcwhile-id										
Organization Artifact Kind	**AR-Component-THEN**										
OAK Relation Kind	**then component part of if component**	ARCC01									
OAK Element Kind	arct-id	ARCT01									
Organization Artifact Kind	**AR-Component-ELSE**										
OAK Relation Kind	**else component part of if component**	ARCC01									
OAK Element Kind	arce-id	ARCE01									
Organization Artifact Kind	**AR-Component-ACTION**		ARCT01				ARCE01				
OAK Relation Kind	**action component part of component**	ARCW01		ARCA02	ARCA03			ARCA05	ARCA06		
OAK Relation Kind	**previous action**										
OAK Relation Kind	**fact creation**										
OAK Element Kind	precedence type		FIRST	FIRST	AFTER		FIRST	FIRST	AFTER		
OAK Element Kind	action type	FLOW	BLOCK	ATOMIC	ATOMIC		BLOCK	ATOMIC	ATOMIC		
OAK Element Kind	atomic action type			PRODUCE_DATA	C-Act			PRODUCE_DATA	C-Act		
OAK Element Kind	action description			the final charge of [rental] = [money]	rental payment of [rental] must be requested			the final charge of [rental] = [money] + [total_of_penalties]	rental payment of [rental] must be requested		
OAK Element Kind	generic requirement										
OAK Element Kind	implementation requirement										
OAK Element Kind	arca-id	ARCA01	ARCA02	ARCA03	ARCA04		ARCA05	ARCA06	ARCA07		

Figure 146: EU-Rent - Action rule 8 - UEAOM instantiation

Organization Artifact Kind	**Action Rule**			
OAK Relation Kind	**executing actor**	B-A02 - Rental Ender		
OAK Element Kind	action rule id	AR09		
Organization Artifact Kind	**AR-Component-WHEN**			
OAK Relation Kind	**when component part of action rule**	AR09		
OAK Element Kind	arcw-id	ARCW01		
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**			
OAK Relation Kind	**enacting transaction component part of when component**	ARCW01		
OAK Relation Kind	**enacting transaction**	T05 - Rental payment		
OAK Element Kind	enacting transaction C-fact	stated		
OAK Element Kind	arcet-id	ARCET01		
Organization Artifact Kind	**AR-Component-CONDITION**			
OAK Relation Kind	**subcondition of condition**			
OAK Element Kind	condition type	Expression		
OAK Element Kind	expression	everything is ok		
OAK Element Kind	arcc-id	ARCC01		
Organization Artifact Kind	**AR-Component-IF**			
OAK Relation Kind	**flow component of action**	ARCA01		
OAK Relation Kind	**evaluated condition**	ARCC01		
OAK Element Kind	arci-id	ARCIF01		
Organization Artifact Kind	**AR-Component-FOREACH**			
OAK Relation Kind	**flow component of action**			
OAK Relation Kind	**type of element**			
OAK Element Kind	arcf-id			
Organization Artifact Kind	**AR-Component-WHILE**			
OAK Relation Kind	**flow component of action**			
OAK Relation Kind	**evaluated condition**			
OAK Element Kind	arcwhile-id			
Organization Artifact Kind	**AR-Component-THEN**			
OAK Relation Kind	**then component part of if component**	ARCIF01		
OAK Element Kind	arct-id	ARCT01		
Organization Artifact Kind	**AR-Component-ELSE**			
OAK Relation Kind	**else component part of if component**	ARCIF01		
OAK Element Kind	arce-id	ARCE01		
Organization Artifact Kind	**AR-Component-ACTION**			
OAK Relation Kind	**action component part of component**	ARCW01	ARCT01	ARCE01
OAK Relation Kind	**previous action**			
OAK Relation Kind	**fact creation**			
OAK Element Kind	precedence type		FIRST	FIRST
OAK Element Kind	action type	FLOW	ATOMIC	ATOMIC
OAK Element Kind	atomic action type		C-Act	C-Act
OAK Element Kind	action description		rental payment of [rental] must be accepted	rental payment of [rental] must be rejected
OAK Element Kind	generic requirement			
OAK Element Kind	implementation requirement			
OAK Element Kind	arca-id	ARCA01	ARCA02	ARCA03

Figure 147: EU-Rent - Action rule 9 - UEAOM instantiation

Organization Artifact Kind	**Action Rule**			
OAK Relation Kind	**executing actor**	B-A02 - Rental Ender		
OAK Element Kind	action rule id	AR10		
Organization Artifact Kind	**AR-Component-WHEN**			
OAK Relation Kind	**when component part of action rule**	AR10		
OAK Element Kind	arcw-id	ARCW01		
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**			
OAK Relation Kind	**enacting transaction component part of when component**	ARCW01		
OAK Relation Kind	**enacting transaction**	T05 - Rental Payment		
OAK Element Kind	enacting transaction C-fact	accepted		
OAK Element Kind	arcet-id	ARCET01		
Organization Artifact Kind	**AR-Component-CONDITION**			
OAK Relation Kind	**subcondition of condition**			
OAK Element Kind	condition type			
OAK Element Kind	expression			
OAK Element Kind	arcc-id			
Organization Artifact Kind	**AR-Component-IF**			
OAK Relation Kind	**flow component of action**			
OAK Relation Kind	**evaluated condition**			
OAK Element Kind	arci-id			
Organization Artifact Kind	**AR-Component-FOREACH**			
OAK Relation Kind	**flow component of action**			
OAK Relation Kind	**type of element**			
OAK Element Kind	arcf-id			
Organization Artifact Kind	**AR-Component-WHILE**			
OAK Relation Kind	**flow component of action**			
OAK Relation Kind	**evaluated condition**			
OAK Element Kind	arcwhile-id			
Organization Artifact Kind	**AR-Component-THEN**			
OAK Relation Kind	**then component part of if component**			
OAK Element Kind	arct-id			
Organization Artifact Kind	**AR-Component-ELSE**			
OAK Relation Kind	**else component part of if component**			
OAK Element Kind	arce-id			
Organization Artifact Kind	**AR-Component-ACTION**			
OAK Relation Kind	**action component part of component**	ARCW01		
OAK Relation Kind	**previous action**		ARCA01	ARCA02
OAK Relation Kind	**fact creation**			
OAK Element Kind	precedence type	FIRST	FIRST	FIRST
OAK Element Kind	action type	BLOCK	ATOMIC	ATOMIC
OAK Element Kind	atomic action type		C-Act	C-Act
OAK Element Kind	action description		rental end of [rental] must be executed	rental end of [rental] must be stated
OAK Element Kind	generic requirement			
OAK Element Kind	implementation requirement			
OAK Element Kind	arca-id	ARCA01	ARCA02	ARCA03

Figure 148: EU-Rent - Action rule 10 - UEAOM instantiation

Organization Artifact Kind	**Action Rule**	
OAK Relation Kind	**executing actor**	B-A02 - Rental Ender
OAK Element Kind	action rule id	AR11
Organization Artifact Kind	**AR-Component-WHEN**	
OAK Relation Kind	**when component part of action rule**	AR11
OAK Element Kind	arcw-id	ARCW01
Organization Artifact Kind	**AR-Component-ENACTING TRANSACTION**	
OAK Relation Kind	**enacting transaction component part of when component**	ARCW01
OAK Relation Kind	**enacting transaction**	T02 - Rental End
OAK Element Kind	enacting transaction C-fact	stated
OAK Element Kind	arcet-id	ARCET01
Organization Artifact Kind	**AR-Component-CONDITION**	
OAK Relation Kind	**subcondition of condition**	
OAK Element Kind	condition type	
OAK Element Kind	expression	
OAK Element Kind	arcc-id	
Organization Artifact Kind	**AR-Component-IF**	
OAK Relation Kind	**flow component of action**	
OAK Relation Kind	**evaluated condition**	
OAK Element Kind	arci-id	
Organization Artifact Kind	**AR-Component-FOREACH**	
OAK Relation Kind	**flow component of action**	
OAK Relation Kind	**type of element**	
OAK Element Kind	arcf-id	
Organization Artifact Kind	**AR-Component-WHILE**	
OAK Relation Kind	**flow component of action**	
OAK Relation Kind	**evaluated condition**	
OAK Element Kind	arcwhile-id	
Organization Artifact Kind	**AR-Component-THEN**	
OAK Relation Kind	**then component part of if component**	
OAK Element Kind	arct-id	
Organization Artifact Kind	**AR-Component-ELSE**	
OAK Relation Kind	**else component part of if component**	
OAK Element Kind	arce-id	
Organization Artifact Kind	**AR-Component-ACTION**	
OAK Relation Kind	**action component part of component**	ARCW01
OAK Relation Kind	**previous action**	
OAK Relation Kind	**fact creation**	
OAK Element Kind	precedence type	FIRST
OAK Element Kind	action type	ATOMIC
OAK Element Kind	atomic action type	C-Act
OAK Element Kind	action description	rental end of [rental] must be accepted
OAK Element Kind	generic requirement	
OAK Element Kind	implementation requirement	
OAK Element Kind	arca-id	ARCA01

Figure 149: EU-Rent - Action rule 11 - UEAOM instantiation