INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Resources Events Agents (REA), a text DSL for OMNIA Entities

MANUEL PAULO MATA
Outubro de 2018

POLITÉCNICO
DO PORTO

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Resources Events Agents (REA), a text DSL for OMNIA Entities

MANUEL PAULO MATA
Outubro de 2018

POLITÉCNICO
DO PORTO

Resources Events Agents (REA), a text DSL for OMNIA Entities

Paulo Mata

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Software Engineering

Supervisor: Professor Alexandre Bragança

Evaluation Committee:
President:
Dr.

Members:
Dr.
Dr.
Dr.

Porto, October 14, 2018

Dedicatory

For all my family, especially my wife Cristina, my daughter Catarina and son João.

Abstract

The Numbersbelieve has been developing the OMNIA platform. This is a web application platform for developing applications using Low-code principles, using Agile approaches. *Modeling Entities* is an application that is used on the platform to create new entities. The OMNIA *Entity* concept has the following properties: *Agents, Commitments, Documents, Events, entities, Resources or Series*. Most of these concepts are in accordance with the Resources Events Agents (REA) ontology but are not formalized. One of the goals of Numbersbelieve is a formalization of the REA concepts according to the ontology for the application that creates entities on OMNIA platform and later for other applications.

REA defines an enterprise ontology developed by McCarthy (1979, 1982) has its origin in accounting database systems. Later Geerts and McCarthy (2002, 2006) extended the original model with new concepts.

To formalize the concepts of the REA ontology, this research shows the development of a textual Domain-Specific Language (DSL) based on the development methodology Model Driven Engineering (MDE) which focuses software development on models. This simplifies the engineering processes as it represents the actions and behaviors of a system even before the start of the coding phase.

This research is structured according to the Design Science Research Methodology (DSRM). The Design Science (DS) is a methodology for solving problems that seek to innovate by creating useful artifacts that define practices, projects and implementations and is therefore suitable for this research. This research developed three artifacts for the formalization of the DSL, a meta-model the abstract syntax, a textual language the concrete syntax and a Json file for interaction with OMNIA.

The first phase of DSRM was to identify the problem that was mentioned above. The following focuses on the identification of requirements which identified the REA concepts to be included in the meta-model and textual language. Subsequently, the development of the artifacts and the editor of the language. The editor allows *use cases*, provided by the Numbersbelieve team, to be defined with the DSL language, correct faults and improve the language. The results were evaluated according the objectives and requirements, all successfully completed.

Based on the analysis of the artifacts, the use of the language and the interaction with the OMNIA platform, through the Json file, it is concluded that the use of the DSL language is suitable to interact with the OMNIA platform through the Application Program Interface (API) and helped demonstrate that other applications on the platform could be modeled using a REA approach.

Keywords: REA, DSL, Meta-Modeling, MDE, DSRM, OMNIA

Resumo

A Numbersbelieve tem vindo a desenvolver a plataforma OMNIA. Esta plataforma é uma aplicação web para o desenvolvimento de aplicações usando princípios Low-code, usando abordagens Agile. *Modeling Entities* é a aplicação que é usada na plataforma para criar novas entidades. O conceito OMNIA de *Entidade* tem as seguintes propriedades:*Agents, Commitments, Documents, Events, Generic entities, Resources or Series*. A maior parte destes conceitos estão de acordo com a ontologia REA mas não estão formalizados. Um dos objetivos da Numbersbelieve é ter uma formalização dos conceitos REA de acordo com a ontologia para a aplicação que cria as entidades na plataforma OMNIA e posteriormente para as outras aplicações.

REA define uma ontologia empresarial desenvolvida por McCarthy (1979, 1982) tem sua origem nos sistemas de base de dados para contabilidade. Mais tarde Geerts and McCarthy (2002, 2006) estenderam o modelo original com novos conceitos.

Para formalizar os conceitos da ontologia REA, esta pesquisa mostra o desenvolvimento de uma DSL textual com base na metodologia de desenvolvimento MDE que foca o desenvolvimento de software no modelo. Esta simplifica os processos de engenharia pois representa as ações e comportamentos de um sistema mesmo antes do início da fase de codificação.

A pesquisa está estruturada de acordo com a DSRM. O DS é um metodologia para resolver problemas que procuram inovar criando artefactos úteis que definem práticas, projetos e implementações e por isso é adequado a esta pesquisa que desenvolveu três artefactos para a formalização da DSL, um meta-modelo a sintaxe abstrata, uma linguagem textual a sintaxe concreta e uma ficheiro Json para interação com a plataforma OMNIA.

A primeira fase do DSRM foi identificar o problema que foi referido em cima. A seguinte concentra-se na identificação dos requisitos que identificaram os conceitos REA a serem incluídos no meta-modelo e na linguagem textual. Posteriormente, é feito o desenvolvimento dos artefactos e do editor da linguagem. O editor permite definir, com a DSL, os *casos de uso* fornecidos pela equipa da Numbersbelieve, corrigir falhas e melhorar a linguagem. Os resultados foram avaliados de acordo com o cumprimento dos requisitos. Foram todos foram concluidos com exito.

Com base na análise dos artefactos, do uso da linguagem e da interação com a plataforma OMNIA, através do ficheiro Json, conclui-se que a utilização da linguagem é adequada para interagir com a plataforma OMNIA através da sua API e ajudou a demonstrar que outras aplicações da plataforma podem ser modeladas usando uma abordagem REA.

Palavras-chave: REA, DSL, Meta-Modeling, MDE, DSRM, OMNIA

Acknowledgement

My thanks to the ISEP institution, to Professor Alexandre Bragança for his patience, to the entire team and to Carlos Morais CEO of Numbersbelieve for all the availability they have always shown. I end by thanking Cristina for her patience and all her support without which this work would not be possible.

Contents

List of Figures

List of Acronyms

ANTLR	ANother Tool for Language Recognition.
API	Application Program Interface.
ARIS	Architecture of Integrated Information Systems.
AS	Abstract Syntax.
ASG	Abstract Syntax Graph.
AST	Abstract Syntax Tree.
BMO	Business Model Ontology.
BNF	Backus–Naur Form.
CCM	CORBA Components Model.
CS	Concrete Syntax.
DBMS	Database Management System.
DI	Dependency Injection.
DS	Design Science.
DSL	Domain-Specific Language.
DSRM	Design Science Research Methodology.
DSVL	Domain Specific Visual Language.
EBNF	Extended Backus–Naur Form.
EMF	Eclipse Modeling Framework.
FEI	Front End of Innovation.
FFE	Fuzzy Front End.
GDPR	EU General Data Protection Regulation.
GMF	Graphical Modeling Framework.
GPL	General-purpose Programming Language.
GUI	Graphic User Interface.
HTML	Hyper Text Markup Language.
HUTN	Human-Usable Textual Notation.
IDE	Integrated Development Environment.
IDL	Interface Definition Language.
IS	Information System.
IT	Information Technology.
JDK	Java Development Kit.

JDT	Java Development Tooling.
JET	Java Emitter Templates.
JMerge	Java Merge.
JVM	Java Virtual Machine.
LSP	Language Server Protocol.
MDA	Model Driven Architecture.
MDD	Model Driven Development.
MDE	Model Driven Engineering.
MDSD	Model Driven Software Development.
MOF	Meta-Object Facility.
MPS	Meta Programming System.
MSDK	Modeling SDK for Visual Studio.
NCD	New Concept Development Model.
NPPD	New Product and Process Development.
OCL	Object Constraint Language.
OMG	Object Management Group.
PIM	Platform Independent Models.
QVT	Queries Views and Transformations.
RAD	Rapid Application Development.
RCP	Rich Client Platform.
REA	Resources Events Agents.
SQL	Structured Query Language.
TEF	Textual Editing Framework.
TOVE	Toronto Virtual Enterprise.
UML	Unified Modeling Language.
XMI	XML Meta-data Interchange.
XML	Extensible Markup Language.

Chapter 1

Introduction

This chapter is dedicated to the presentation of the context and the background of this thesis and the problem in analysis. It presents the objectives, research methodology used and describes the content developed in the following chapters of this research document.

1.1 Context

Numbersbeleave has been developing OMNIA platform, which is a Web application for Agile development and deployment of applications using Low-Code principles [1]. Modeling Entities is the application that is used by platform users to create and define new entities, based on the Entity OMNIA concept (Agents, Commitments, Documents, Events, Generic entities, Resources or Series) [2]. Most of them are Resources Events Agents (REA) concepts however they are not defined according to the ontology. NumbersBelieve aims to provide the OMNIA platform with a Domain-Specific Language (DSL) formalized on REA concepts.

REA defines an enterprise ontology developed by McCarthy (1979, 1982) has its origin in accounting database systems. Later Geerts and McCarthy (2002, 2006) extended the original model with new concepts.

This masters research provides a series of artifacts to model the OMNIA entities based on REA ontology and means to support the current model. The artifacts are: meta-model and a textual DSL based on REA core concepts, a JSON file created using code generation of an entity instance compatible with the existing OMNIA model, that can be used to automatically create, through existing API, an OMNIA application. It is expected that in the future the model of entities in the OMNIA will be based on formalized DSL.

1.2 Problem Description

The OMNIA platform defines an Entity with the concepts *Agents, Commitments, Documents, Events, Generic entities, Resources or Series*. Four of the six concepts are REA concepts (Resources, Events, Agents and Commitments) however they are not defined according to the ontology. Some examples not in accordance are: in the events is not defined the type (Increment or Decrement) there are commitments without contract and for the agents is also not defined the type (external or internal). Numbersbeleave wants to have

[1] `https://docs.numbersbelieve.com/index.html`
[2] `https://docs.numbersbelieve.com/omnia3_modeler_entities.html`

a DSL formalized and supported in solid theoretical concepts base on REA ontology. The formalization of the DSL can make possible to verify in the existing activities some activity that does not make sense, now. Which can help to improve the design of the platform. Once the textual DSL is developed it is also possible to create a stand-alone editor and provide it to clients as a complementary tool for interacting with the platform, due to the implementation of semantic and syntactic verification the output data is free of this type of errors.

1.2.1 Research Question

Although the OMNIA platform defines REA concepts they do not conform to the formal proposal of the ontology. For formalization it is necessary to create the necessary artifacts. The first step is to create a meta-model that defines an abstract syntax, and a textual language that defines a concrete syntax that at the same time serves as an interface for users. However it is necessary to maintain compatibility with what already exists and for this is defined a means of interaction through the automatic generation of the necessary artifacts. This work will help answer the question:

> *"How to formalize a textual DSL based on the REA ontology and at the same time be compatible with the existing model and concepts in the OMNIA platform?"*

1.3 Objectives

The objectives of this thesis are as follows:

- Define a meta-model (abstract syntax) DSL of REA.
- Define a textual (concrete syntax) DSL of REA.
- Transform an instance of the created model to be compatible with the current model. The JSON file created, using automatic code generation, is the output. The file is the interaction medium with the OMNIA platform.

1.4 Approach and Development Process

In the Information System (IS) discipline, two paradigms characterize most of the research. Behavioral Science and Design Science. (Hevner et al. 2004) Behavioral Science is used to develop and justify theories such as principles and laws. Design Science (DS) is a paradigm for solving problems seeking to innovate by creating useful artifacts defining practices, designs and implementations (Hevner et al. 2004). The research in this thesis aims to build and test some artifacts to solve a problem, so it can be characterized as DS. Design Science research can be carried out following different frameworks. According to (Hevner et al. 2004) and (Peffers et al. 2007) Design Science Research Methodology (DSRM) is widely accepted as design science methodology. This methodology consists mainly of six activities, according to (Peffers et al. 2007), defined as follows:

1. *Problem identification and motivation:* Define the problem under study and justify the value of a solution. While the problem definition will be used to develop the artifact that can actually solve the problem. The justification of the value of the solution essentially aims at two things: To motivate the researcher to search for the solution and to accept the results. (Peffers et al. 2007)

   The definition of the problem has already been made in section 1.2.

   From the analysis of the literature it was found that a textual DSL of the REA ontology was not found. This research revolves around the formalization of a textual DSL based on the REA ontology for the OMNIA platform.

2. *Define the objectives for a solution:* Objectives must be inferred from the specification of the problem and can be quantitative or qualitative. Knowledge of the state of the problem and existing solutions, if any, and their effectiveness. (Peffers et al. 2007)

   The objectives are defined in section 1.3 according to the problems in section 1.2.

3. *Design and development:* Determine the functionalities of the artifact and their architecture and finally create the artifact.(Peffers et al. 2007)

   The design process will address how to create a meta-model, a textual syntax of REA DSL and an JSON file, created through code generation, used as a means of interaction with the OMNIA platform.

4. *Demonstration:* Demonstrate the use of the artifact to solve one or more instances of the problem. The use of experimentation, simulation, case study or any other appropriate activity may be required.(Peffers et al. 2007)

   Several cases of use of the current model, cases described in the literature and an JSON file used as a means of interaction with the OMNIA platform will be used as demonstration.

5. *Evaluation:* Observe and measure how the artifact support the solution to the problem. It is necessary to compare the results observed in the demonstration with the initially established objectives.(Peffers et al. 2007)

   The evaluation is done according to the percentage of completed objectives. The effective interaction between artifacts created and the system being analyzed. The elements of the company's team will evaluate quality variables defined together.

6. *Communication:* Communicate all relevant aspects of the problem, the artifact and its usefulness and effectiveness to practicing professionals and the community in general. This research is a master's thesis the report itself is the means of communication for the community.(Peffers et al. 2007)

1.5 Document Structure

This document is divided into six chapters. The first chapter introduces the theme of the thesis, the context, the description of the problem, defines the objectives, approach and process of development and the structure of the document. The second chapter shows a general context, Low-code platforms, OMNIA platform, REA ontology, existing REA DSLs, Model Driven Engineering (MDE) and DSL technologies. The third chapter performs value

analysis, value preposition and CANVAS. The fourth chapter shows DSL analysis and design. The fifth chapter details the implementation and evaluates the artefact's. In the last chapter conclusion, limitations and future work.

Chapter 2

Context and State of the Art

This chapter provides a historical introduction to Rapid Application Development (RAD) applications, describes the concept of the Low-code Platform, the OMNIA platform, defines ontology and the REA ontology, describes DSL concept, makes a literature-based description of existing REA DSLs, MDE and DSL technologies, and finally justifies the choice of tools used in the development of the artifacts of this thesis.

2.1 Context

In the 1990s, approaches to the development of applications that at the time could be called "Low-code" were named RAD. A set of tools that had great popularity, Microsoft Access, Visual Basic, Delphy and PowerBuilder, are examples of such applications. These platforms became popular in response to the emergence of personal computers with Graphic User Interface (GUI)s that could be networked. The rapid expansion of operating systems and client-server Database Management System (DBMS)s has put Information Technology (IT) departments under great pressure to develop applications that use this new technology. The challenge was to develop applications with traditional tools that would run everywhere. Today, new computing platforms revolve around cloud-based applications, virtual server platforms, and client mobile applications. Organizations need Low-code platforms for a variety of reasons ranging from the need to deliver more complex applications very quickly that work in heterogeneous environments, distributed environments, and in a tight competition market. For organizations, it is imperative that technology platforms support true collaboration between technicians and other people in the business, enabling these groups to work together quickly and efficiently.(Hilwa 2017)

2.2 Low-code Platforms

Low-code platforms benefit the development of complex applications faster and with less effort than using the traditional process to develop these applications. This is typically accomplished using a combination of visual tools, model-based development, which allows specify application behavior through high-level configurations rather than code, automatic code generation or interpretation to create the application behind the scenes. (Hilwa 2017)

Most Low-code platforms are built around three distinct layers:

- A layer for data management (where the data to be managed is specified)

- The business logic layer (where the main behavior of the application is specified)
- A layer focused on user experience (how they interact with the application, screens, forms, menus and so on)(Hilwa 2017)

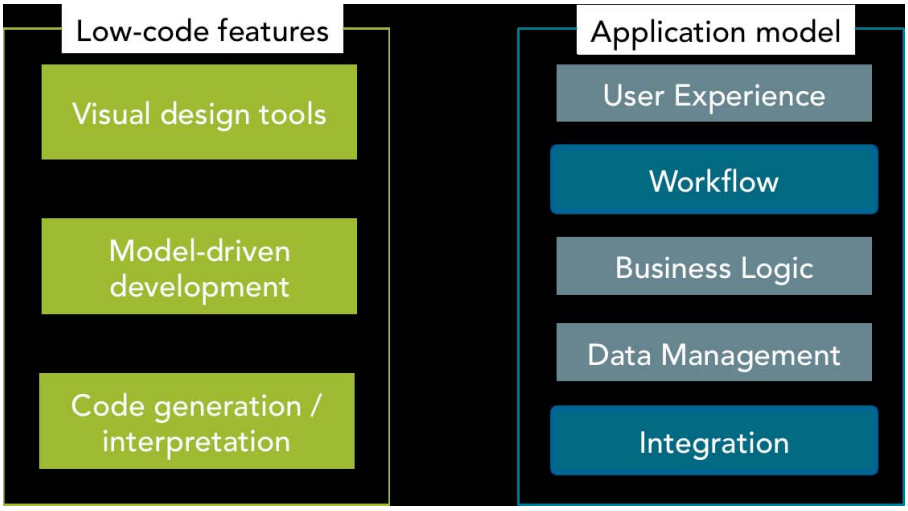


Figure 2.1: Low-code platforms: speed, collaboration, agility - Source:(MWD Advisors)

The applications developed in these platforms are to be put into production and remain in production for several years, so the application life-cycle support is another very important area. So testing applications, deployment, maintenance, advanced design, and requirements gathering are becoming important. Another important factor of differentiation may be the degree of interoperability allowed with other applications, other tools, and other platforms (e.g., Java or .NET). This can be crucial terms of reusing valuable components developed in these platforms (Hilwa 2017).

According to Forrester Research, low-code development platforms can be defined as:

"...enable rapid application delivery with minimal hand-coding,and quick setup and deployment."

Forrester divides the low-code platforms into functional areas, Database, Request-handling, Mobile-first, Process, and General-purpose . The report also shows how Low-code platforms are placed in the market according to the criteria, current offering, strategy and market presence (Rymer 2017).

2.2.1 OMNIA

Numbersbelieve is developing the OMNIA by implementing several of these Low-Code principles. The interaction with the platform can be done in two ways through an API or via web browser with a user interface. It is developed in the .NET framework and uses the C# programming language. It has a multi-tenant architecture, this model provides users and applications with a common and unique infrastructure and code base. Application development and deployment can be tracked in real time by customers and deployed on any cloud-based system provider. All authorizations use a security policy-based model, the image below shows key security concepts. To allow connection to external systems (e.g. client

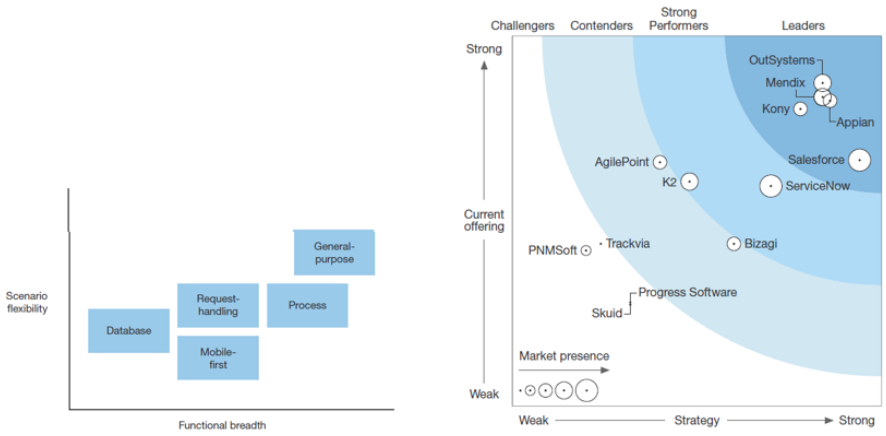


Figure 2.2: Forrester Wave - Low-Code Development Platforms For AD&D Pros

servers) the *Connector* concept is used. The *Management* application allows the management of the various components of the platform, tenants, roles, policies, API clients and connectors. The platform was developed in compliance with the EU General Data Protection Regulation (GDPR) Regulation which allows organizations, that use OMNIA, to also protect the privacy of their customers. Using Forrester functional division, the platform is placed within the area database.

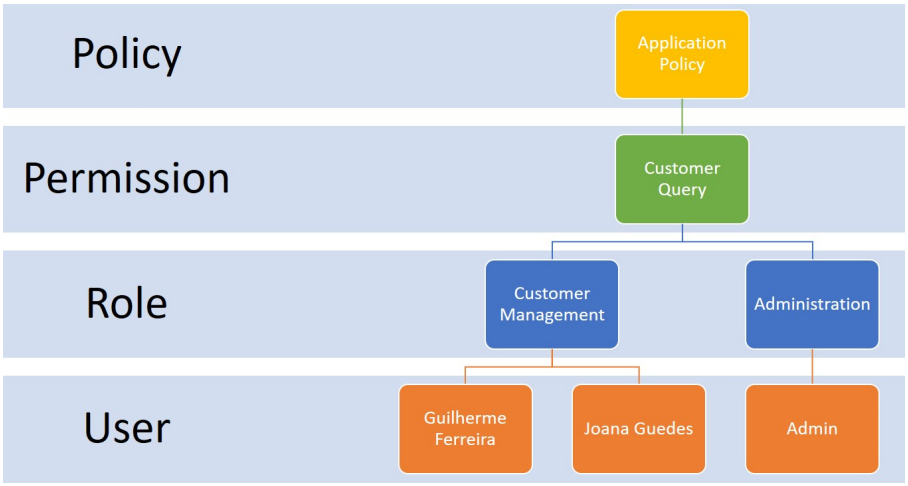


Figure 2.3: OMNIA - Policy-based model three main concepts - source:(Numbersbelieve 2018)

2.3 REA Ontology

The term "Ontology" has evolved and has several definitions according to the area of interest that goes from philosophy, metaphysics and information technologies. (Gruber 1993) indicate that *"an ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the*

vocabulary.", combining several sources it is possible to write that an ontology describes a domain with a semantic that is the basis for understanding that domain. Examples of enterprise ontologies include Architecture of Integrated Information Systems (ARIS), Toronto Virtual Enterprise (TOVE), SAP and REA (O'Leary 2010), Business Model Ontology (BMO) and e3-value ontology (Sonnenberg et al. 2011).

The REA model was published in 1982 by McCarthy. During the 1990s its initial structure was expanded several times as a result of the work of Geerts and McCarthy (1992, 1994, 1997a, b, 1999, 2000a, b). Its initial structure dedicated to accounting is now the framework for enterprise information architectures (Guido L Geerts and Mccarthy 2000). In all enterprise software applications concepts are always present, they derive from knowledge of the domain, and understanding these concepts helps to understand how the application works. Designing a business application without understanding the concepts brings more difficulty and almost any change in requirements will bring many changes in the architecture. The fundamental concepts of REA are *economic resource, economic agent, economic event, contract, and commitment* (Hruby 2013). The fundamental concepts of the REA ontology

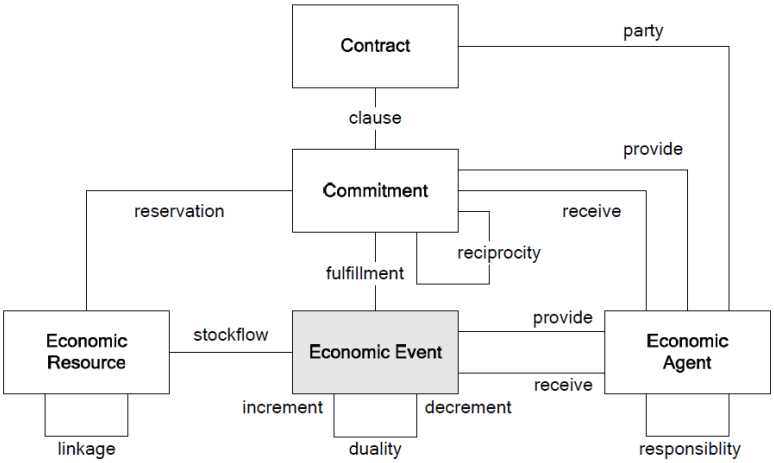


Figure 2.4: Fundamental REA concepts -source:(Hruby, P. 2013)

are described below.

- *Economic Resource* - is a scarce resource, something that needs to be planned, monitored and controlled. Some examples of economic resource are products, services, money, raw materials, labor, tools and services used by a company.(Hruby 2013)
- *Economic Agent* - is an individual or an organization that holds control over an economic resource, which transfers or receives control to or from other individuals or organizations. Examples of economic agent are customers, sellers, employees and organizations. The REA model is created in the perspective of the economic agent *enterprise.*(Hruby 2013)
- *Economic Event* - represents an increment or a decrement in the value of an economic resource that is over control of an enterprise. While some economic events occur instantaneously, like sale of goods, others occur over time like rentals, labor, provision and services.(Hruby 2013)
- *Contract* - is a collection of commitments of the type increment and decrement and its terms. The contract can specify what should happen if the commitments are not met, usually a type of penaltie.(Hruby 2013)

- *Commitment* - promise or obligation of economic agents to carry out an economic event in the future, the items lines in a sales order represent the commitment to sell the goods. (Hruby 2013)

Geerts and McCarthy (1999,2000) *"...rules exist that restrict the use of the REA primitives for the conceptualization of economic phenomena. The recognition and explicit definition of these rules or axioms is an important part of ontological engineering."* These axioms specify domain rules and complement the primitives of REA. Axiom is defined in Bahrami (1999)(Guido L Geerts and Mccarthy 2000) as:*"the fundamental truth that is always observed to be valid and for which there is in the counterexample or exception"*. Geerts and McCarthy (1999,2000) defined three axioms that are part of the REA ontology:

- *First Axiom - "At least one inflow event and one outflow event exist for each economic resource; conversely inflow and outflow events must affect identifiable resources."*(Guido L Geerts and Mccarthy 2000). One *Increment Event* (Inflow) and one *Decrement Event* (Outflow) must exist for each exchange which implies that Economic agents will also exchange the control of the resources (e.g. When a pizzeria sells a pizza loses control of the pizza to the buyer and the buyer loses control of an amount of money).
- *second Axiom - "All events effecting an outflow must be eventually paired in duality relationships with events effecting an inflow and vice-versa."* (Guido L Geerts and Mccarthy 2000). An *Increment Event* must be linked to a *Decrement Event* through a duality relation and vice-versa .
- *third Axiom - "Each exchange needs an instance of both the inside and outside subsets."*(Guido L Geerts and Mccarthy 2000). In commercial transactions both parties (e.g. seller and buyer) must exist as economic agents .

Nonetheless, REA can be extended through a set of business standards that gives it the functionality to develop applications that require specific needs. However REA alone, forms the backbone of the application model, as it specifies a set of domain rules ensuring that the application model derives from the business perspective.(Hruby 2013)

2.4 DSL

A DSL is a language that solves a problem specific to a particular domain. This type of language differs from General-purpose Programming Language (GPL)s, such as C, C++ , C#, Java, Perl or Ruby, which can be applied to solve problems in any domain. Because DSL languages provide abstractions only for one domain problem, it results in greater expressiveness and ease of use compared with GPLs. Some examples of DSL's are Structured Query Language (SQL) (for queries and database manipulation),Hyper Text Markup Language (HTML) (for hypertext and web pages), Backus–Naur Form (BNF) (for syntax specification) or LATEX (for Typesetting). Domain experts understand the definition of DSL because it exposes the knowledge of that domain. Even though it is non-technical the expert uses and changes the DSL to contain domain declarations that can be understood by an information system. The image below shows the main DSL artifacts and their relationships.(Strembeck and Zdun 2009)

The *Language Model* specifies the domain elements to abstract syntax. It is a model composed of three sub-models:

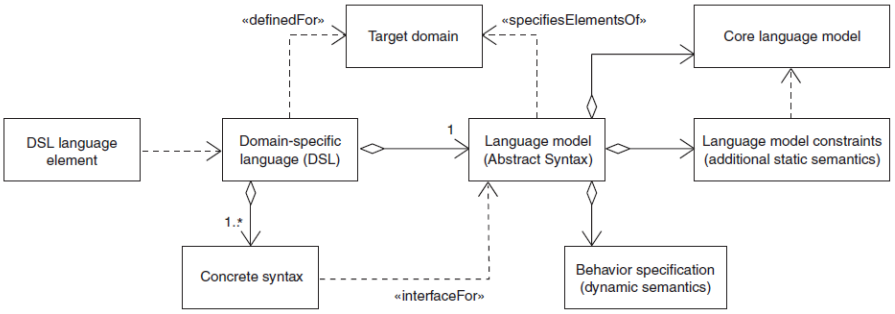


Figure 2.5: Domain-specific languages: artifacts - source:(Strembeck and Zdun, 2009)

- *Core Language Model "...captures all relevant domain abstractions and specifies the relations between these abstractions."* (Strembeck and Zdun 2009). Examples of domain abstractions in REA, resource, event, agent, contract, commitment. More general examples customer , account, address, purchase order. In test software examples are case, test result, test. It is the formalization of concrete knowledge of the domain (Strembeck and Zdun 2009). *"Depending on the type of DSL, the core language can either be a meta-model or an ordinary model"* (Strembeck and Zdun 2009).
- *The Language Model Constraints* which is also referred to as static semantics that can not be represented in the graphical model. For example prevent repeating the name of a property, or force the name of the property to begin with a capital letter, do not allow key words (if, then, else, and, or not). Constraints can be deployed using for example the Object Constraint Language (OCL).(Strembeck and Zdun 2009)
- *The DSL Behavior Specification "sometimes also referred to as dynamic semantics, is a part of the language model and defines the (behavioral) effects that result from using a DSL language element.".*(Strembeck and Zdun 2009)

Having the artifacts to specify the abstract syntax of a DSL is not enough it is necessary to specify a concrete syntax. From the user's perspective the concrete syntax serves as the DSL interface which should be simple and intuitive. There may be several concrete syntaxes, for example a graphical and a textual syntax. The transformation and the generation of code define the format of transformation from one model to another. With the generation of code it is possible to execute transformations that create artifacts based on the respective platform like Java, C# or Ruby for example, the image below shows the transformation artifacts.(Strembeck and Zdun 2009)

Mark Strembeck and Uwe Zdun (Strembeck and Zdun 2009) divides the DSLs, depending on the type of implementation, into two types:

- *An embedded DSL (also called internal DSL) "is defined as an extension to an existing GPL and uses the syntactic elements of the underlying (host) language."*(Strembeck and Zdun 2009). All host language features are available for the DSL, libraries, frameworks, components, debugger, compiler or interpreter, so typically transformations and extra code generator are not required. Examples of embedded DSL are Rails framework (Ruby) or Eclipse framework (Strembeck and Zdun 2009).

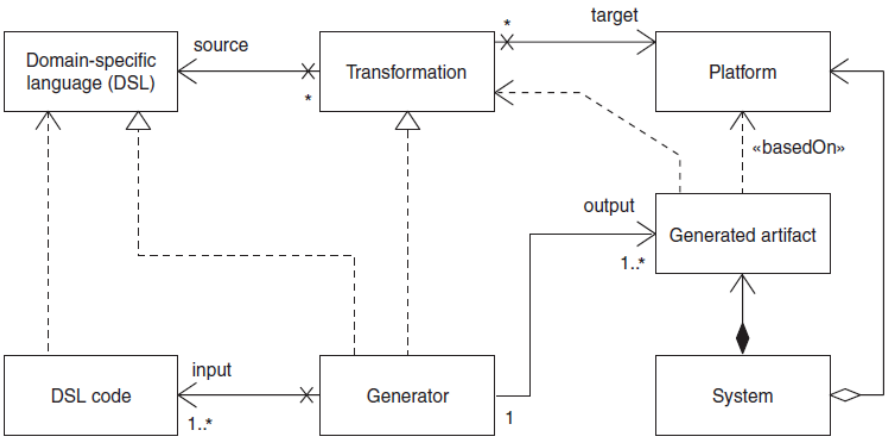


Figure 2.6: DSL transformations: artifacts - source:(Strembeck and Uwe Zdun, 2009)

- *An external DSL "is defined in a different format than the intended target language(s) and can use all kinds of syntactical elements (independent of any other language)."*(Strembeck and Zdun 2009). DSL designers can define textual or graphical syntax without taking into account the language of a host platform. For the user, only the elements exposed by the concrete syntax are available, so it is not possible to inadvertently use features that are not part of the DSL (in embedded DSLs it could happen). Some examples are *XText* framework and *Microsoft OSLO* framework. (Strembeck and Zdun 2009) This is the type of DSL used in this desertion.

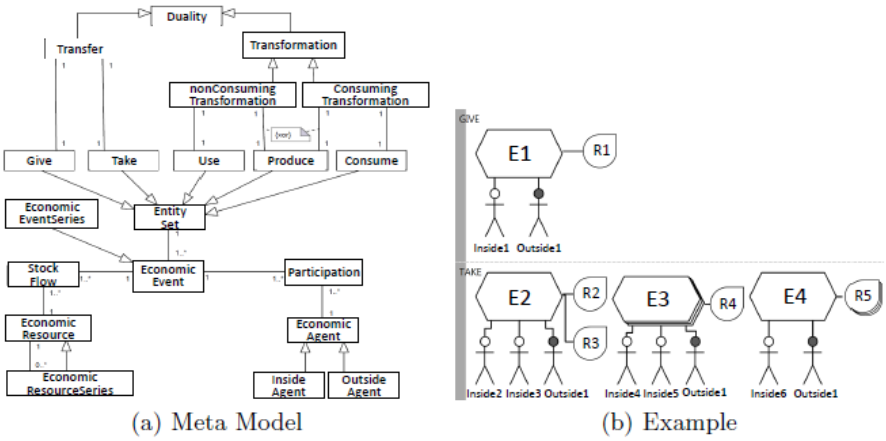
2.5 Existing REA DSLs

This section presents the state of the art of solutions and approaches whose objective is the creation of a DSL based on the REA ontology.

2.5.1 REA - DSL

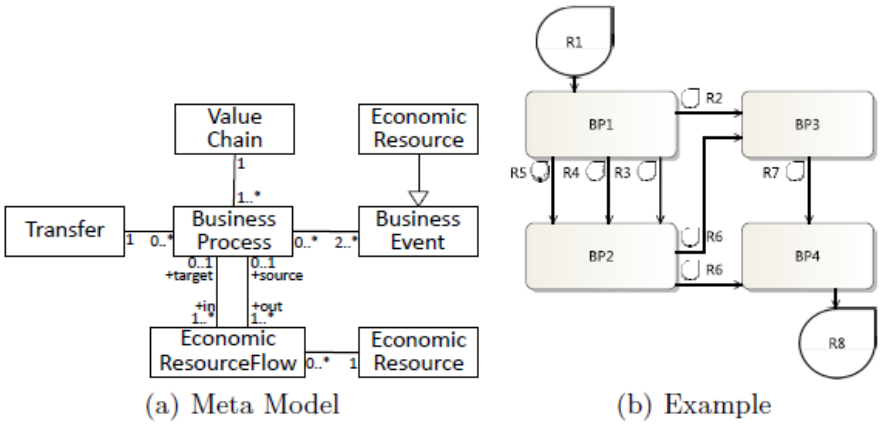
In the conference paper *The REA-DSL: A Domain Specific Modeling Language for Business Models* an Domain Specific Visual Language (DSVL) based on the REA ontology is proposed. In the development of that DSL, the methodological process proposed in *Systematic Development of Domain Species and Languages* (Strembeck and Zdun 2009) was followed. The above development process describes the steps for extracting a DSL from a given domain, in this case the REA domain. Identification of the elements of the REA ontology was done, several revision cycles were processed to identify an abstract syntax, constraints of the model, determine the behavior of DSL and finally the definition of a concrete syntax. Although the resulting meta-model consists of three views, only two are described, the *Duality* and *Value Chain* views. The basis of DSL construction is the Object Management Group (OMG) through its Meta-Object Facility (MOF) architecture, which at its M3 layer implements a meta-model that allows defining the abstract syntax as a M2 layer meta-model (Sonnenberg et al. 2011) .

The images below show (a) the meta-models and (b) examples using the developed tool.



(a) Meta Model (b) Example

Figure 2.7: Duality - source:(Sonnenberg et al. 2011)



(a) Meta Model (b) Example

Figure 2.8: Value Chain - source:(Sonnenberg et al. 2011)

The visual tool, the concrete syntax, was developed using the Microsoft DSL tool kits available in the Visual Studio 2010 Visualization & Modeling SDK. The designer is divided into five areas:*modeling canvas (A), toolbox (B), the solution explorer (C), the property window (D) and the validation window (E)*, the image below shows these areas. A REA graphical model is created by dragging the elements of the toolbox onto the *modeling canvas*. The different instances of the model, the structure of files and directories are shown in *solution explorer* (Sonnenberg et al. 2011) .

In his thesis *REA Business Modeling Language Toward a REA based Domain Specific Visual Language* Mohannad M. Al-Jallad developed the DSVL in four phases: *meta-model development, visual-notation development, visual notation implementation, and editor generation*. For the development of the meta-model the author used the framework described in (SCHAFER, Christian et al., 2011)(Al Jallad 2012), as process guidelines. The tools present in the Graphical Modeling Framework (GMF) based on a model-driven approach provides the foundations necessary for the creation of graphic editors. For modeling the GMF uses the *Ecore* language. These tools are available after installing Eclipse and the necessary plug-ins. Each new GMF project consists of six files whose relationships are described in the image below.(Al Jallad 2012).

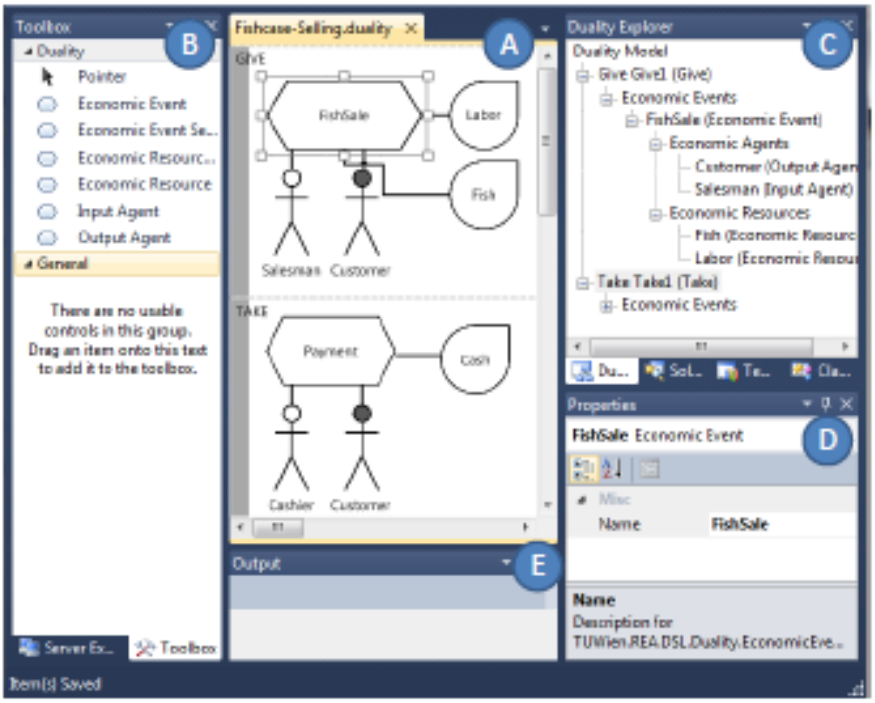


Figure 2.9: REA - Visual Studio designer - source:(Sonnenberg et al. 2011)

File	Purpose
Domain Model	Contains the meta-model Ecore file.
Diagram Gen Model	An auto-generated file based on the meta-model Ecore file. This file is responsible for creating the basic editor environment in the form of Eclipse plug-in.
Graphical Def Model	Contains a file that links the elements of the meta- model with their default or user-defined graphical definitions.
Tooling Def Model	Contains the file that controls the layout of the generated modeling environment.
Mapping Model	Contains the file that links the graphical definitions and the tool functions that will create these elements. Mapping Model is also responsible for defining additional validation rules, and for preparing the environment to be finally released for generation
Diagram Editor Gen Model	is responsible for generating the modeling environment (the editor)

Figure 2.10: GMF Dashboard - source:(Al Jallad 2012)

To develop the graphs that represent the concepts of REA, the author used imagination and creativity and created a set of graphs that implemented as graphic notation. From the two possible options for creating the graphical editor, creating a plugin for eclipse and Rich Client Platform (RCP), the author opted for RCP. The generated application shows an interface with only the menu items for the modeling tasks. The author uses demonstration processes from a company the *ABS Aheels*. One such process is the *Advertising* process, sometimes the company needs to place ads in the press and Internet sites. The description of the process is as follows: "The *company's manager* (Internal agent) pays for the *Advertising channel* (External agent) some *money* (Resource), and the advertising channel publishes *advertisements* (resource) for *ABS Wheels* (Internal agent). For that purpose, an increment exchange event with the name *"Get advertised"* was modeled, then another decrement exchange event with the name *"Pay for advertisements"* was modeled, and both of these events were linked to an exchange duality element".(Al Jallad 2012) The image below shows

the Advertising process that results from using the DSVL editor.

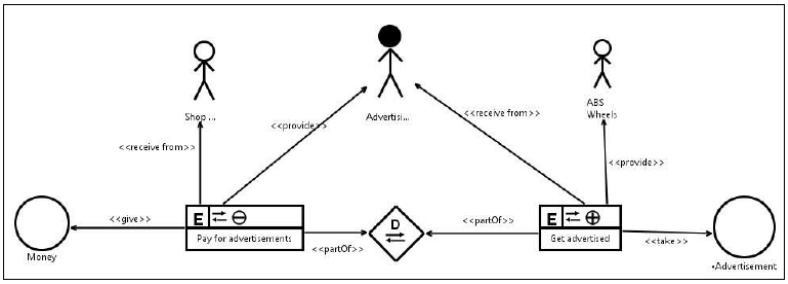


Figure 2.11: ABS Wheels Advertising process - source:((Al Jallad 2012)

The work to represent the REA ontology through DSL or DSVL is limited. The results of the literature review show only the work described above. It was not possible to find works that showed the REA ontology defined as textual DSL.

2.6 MDE Technologies

This section describes some tools associated with the development of design environments that support the Model Driven Development (MDD) approach.

2.6.1 CADENA

CADENA is an integrated environment for building and modeling systems that use CORBA Components Model (CCM). It is possible to define components using CCM Interface Definition Language (IDL), specify dependency information and transitions systems semantics for these types, assembling systems from CCM components, visualizing various dependence relationships between components, specifying and verifying correctness properties of models of CCM systems derived from CCM IDL(Childs et al. 2004) [1] . Some of the capabilities implemented to develop CCM systems are:

- A collection of specifications that can be linked to the IDL.
- Dependency analysis that allows to trace all events.
- A model verification infrastructure (based on Bogor model-checking).
- Java code generation component using OpenCCM.
- A component assembly framework supporting a variety of visualization and programming tools for developing component connections.

[1] `http://cadena.projects.cis.ksu.edu`

2.6.2 Microsoft DSL Tools

Microsoft DSL tools are integrated into Visual Studio through the DslPackage or Modeling SDK for Visual Studio (MSDK). Allows DSL design and generates all that is required to create models based on that DSL.Available tools include: [2]

- A project wizard that uses different solution templates to help you start developing your domain-specific language.
- A graphical designer for creating and editing your domain-specific language definition.
- A validation engine that makes sure that the domain-specific language definition is well-formed, and displays errors and warnings if there are problems.
- A code generator that takes a domain-specific language definition as input and produces source code as output.

By using a specialized editor available with MSDK to define a schema or abstract syntax together with a graphical notation. From this definition, VMSDK generates: [3]

- A model implementation with a strongly-typed Application Program Interface (API) that runs in a transaction-based store.
- A tree-based explorer.
- A graphical editor in which users can view the model or parts of it that you define.
- Serialization methods that save your models in readable XML.
- Facilities for generating program code and other artifacts using text templating.

2.6.3 Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) is a modeling framework in which it uses an Model Driven Architecture (MDA) approach to MDD. MOF is the EMF underlying meta-meta-model. Supported model specifications range from using XML Meta-data Interchange (XMI), java annotated, and the Extensible Markup Language (XML) schema. Generating Java code allows the manipulation of the elements in the model. Many technologies and frameworks are based on EMF (CORE), server solutions, persistence frameworks, UI frameworks and support for transformation. Some of the EMF core technologies are: [4]

- *EMF* - The core EMF framework includes a meta model (Ecore) for describing models and runtime support for the models.
- *EMF.Edit* - he EMF.Edit framework includes generic reusable classes for building editors for EMF models.
- *EMF.Codegen* - The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model.The generation facility leverages the Java Development Tooling (JDT) component of Eclipse.

[2] https://msdn.microsoft.com/en-us/library/bb126327.aspx
[3] https://msdn.microsoft.com/en-us/library/bb126259.aspx
[4] https://www.eclipse.org/modeling/emf/

2.6.4 MOF

MOF 2.5 provides the basis for the definition of meta-model within OMG languages (including UML). The MOF 2 is core is the foundation of the other OMG MOF specifications. The following list shows the characteristics of the various specifications (OMG, 2017 [5]).

- XMI - for interchanging MOF-based models in XML.
- MOF 2 Facility and Object Life-cycle - for connecting to and managing collections of MOF-based model elements
- MOF 2 Versioning and Development Life-cycle - for managing versions and configurations of MOF-based models.
- MOF Queries Views and Transformations (QVT) - for transforming MOF-based models
- MOF Models to Text - for generating text, such as programs, from MOF-based models
- OCL - for specifying constraints on MOF-based models

Meta-data

Meta-data can be defined as information double the data or a set of elements defined by the Unified Modeling Language (UML) so that it is possible to develop models. A model developed in this way, in the hierarchy of the meta-models, becomes meta-data for the application that uses it. The MOF provides an open and independent platform Platform Independent Models (PIM) so that together with meta-data services it facilitates the development of Meta-data-driven systems (OMG, 2017 [6]).

META-MODEL HIERARCHY

The hierarchy of the OMG meta-framework in which the MOF is at the highest level of abstraction, where the items at each level depend on the items at the level above (OMG, 2017 [7]).

M3 MOF	Defines a language for specifying a metamodel Example: MOF
M2 UML	Defines a language for specifying models Example: UML
M1 User Models	Defines a language that describe semantic domains Example: model of a problem domain
M0 Instance Models	Contains run-time instances of the model elements defined in a model

Figure 2.12: OMG Model Hierarchy (OMG, 2017)

[5] Available at: http://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf
[6] Available at: http://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf
[7] Available at: http://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf

ABSTRACT SYNTAX and SEMANTICS

The MOF is described through two textual and graphic presentations. To describe the abstract syntax is a combination of languages, a UML subset, OCL, and natural language. MOF semantics are typically described in natural language. For example, we have the Class Element that is at the top of the hierarchy that is described in the MOF 2.5 specification as:

Class Element is the superclass of all classes defined in MOF, and is an implicit superclass of all meta-classes defined using MOF: this superclass relationship to Element does not need to be explicitly modeled in MOF-compliant meta-models, and if implicit in this way Element is not included in the list of super classes. By creating Properties with type Element it is possible to reference elements in any MOF-compliant model, similar to the use of xsd:any in XML Schema's. Each element can access its meta-class in order to obtain a Class that provides a reflective description of that element. By having both MOF and instances of MOF be rooted in class Element, MOF supports any number of meta layers. (OMG 2017).

2.6.5 MDA

The MDA approach represents and supports almost everything from the modeling of business requirements to the implementation of technology. This is achieved through models that address the complexity of large systems, interaction and collaboration between organizations, people, hardware and software (OMG, 2017 [8]).

2.7 DSL Technologies

A textual and domain-specific language is typically declarative, yet it offers an expressive power with specific abstractions and notations focused on a given domain problem. Domain experts can easily understand and often modify programs / models which improves productivity, reliability and maintenance. They also allow validation and optimization at a domain level (Sonnenberg et al. 2011).

2.7.1 XText

XText is an Eclipse.org project that is available as open-source. It provides a set of APIs and DSLs that allow the development of an DSL from scratch. The support for *Eclipse, IntelliJ*, Web and integration with the build tools *Maven, Gradle and Ant*. XText is a framework for the development of programming languages and specific domain languages. It also provides a complete infrastructure including parser, linker, typechecker, editing support for Eclipse, or any editor that supports the Language Server Protocol (LSP) and web browser. [9]

- The new language produced with XText includes different aspects through a set of DSLs and APIs and then fully implemented in the Java Virtual Machine (JVM). The

[8] Available at: `http://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf`
[9] `https://www.eclipse.org/Xtext/#feature-overview`

compilation is independent of Eclipse and OSGi and can be used in any Java environment. The most important runtime components are: the parser, type-safe abstract syntax tree (AST), the serializer, code formatter, scoping framework, the linking, compiler checks and static analysis aka validation and a code generator or interpreter. Because these components are based on EMF, XText can be used in conjunction with other frameworks also based on EMF such as GMF.9

- XText uses the *Google Guice* framework for Dependency Injection (DI) that integrates the language and the entire Integrated Development Environment (IDE) infrastructure. However it is possible to customize the *Google Guice* through non-evasive changes on several classes. It is possible to build editors for GPL and DSL languages. In the background uses an ANother Tool for Language Recognition (ANTLR) type LL (*) parser, allowing a wide range of syntaxes to be covered the terminal rules are described using Extended Backus–Naur Form (EBNF) expressions.9
- The definition of the grammar of the XText language includes an IDE that adapts automatically to each language produced. Whenever a produced grammar changes, the behavior of the text editor is updated without any change to the code for the features: handling of cross-references, code completion, navigation, syntax coloring and validation. In order for the behavior of the IDE to be adapted to the characteristics of different languages, XText allows you to change the default implementation. As mentioned above this is done through *Google Guice*.9
- The text formats created with XText can be combined with various graphic editing frameworks, for example GEF, Sirius or Graphiti, this flexibility is guaranteed by EMF support.9The image below shows the Features by platform.

	LSP	eclipse	
Syntax Coloring	✓	✓	✓
Semantic Coloring		✓	✓
Error Checking	✓	✓	✓
Auto-Completion	✓	✓	✓
Formatting	✓	✓	✓
Hover Information	✓	✓	✓
Mark Occurences	✓	✓	✓
Go To Declaration	✓	✓	
Rename Refactoring	✓	✓	
Debugging		✓	
Toggle Comments	✓	✓	
Outline / Structure View	✓	✓	
Quick Fix Proposals	✓	✓	
Find References	✓	✓	
Call Hierarchy		✓	
Type Hierarchy		✓	
Folding		✓	

Figure 2.13: Editor Features By Platform - source9

The workflow of XText is as follows: it starts with the definition of Abstract Syntax (AS) and then the definition of Concrete Syntax (CS), this is done in the *.xtext* file. CS is defined as a context-free grammar which includes the terminal symbols and rules of production. The *.mwe (Modeling Workflow Engine)* file describes the steps for: loading the model, executing checkers, and generating code. Through the *.xtext* file is generated Abstract Syntax Graph (ASG) (Ecore file), ANTLR based scanner and parser for DSL-to-model transformation, model-to-text generator with Xpand and DSL editor support created in Eclipse. The editor

supports syntax highlighting, code completion, navigation and reference, folding, bracket matching, styled label providers, and incremental code generation.9

2.7.2 XTend

XText uses XTend in it's infrastructure and also to model transformation and management and code generation. Xtend is a flexible and expressive dialect of Java, which compiles into readable Java 8 compatible source code. You can use any existing Java library seamlessly. [10] It's a statically-typed programming language which translates to Java code. The roots of the language are the java syntax and semantics but better in several aspects: [11]

- *Extension methods* - enhance closed types with new functionality.11
- *Lambda Expressions* - concise syntax for anonymous function literals.11
- *ActiveAnnotations* - annotation processing on steroids.11
- *Operator overloading* - make your libraries even more expressive.11
- *Powerful switch expressions* - type based switching with implicit casts.11
- *Multiple dispatch* - a.k.a. polymorphic method invocation.11
- *Template expressions* - with intelligent white space handling.11
- *No statements* - everything is an expression.11
- *Properties* - shorthands for accessing and defining getters and setter.11
- *Type inference* - you rarely need to write down type signatures anymore.11
- *Full support for Java generics* - including all conformance and conversion rules.11
- *Translates to Java not bytecode* - understand what is going on and use your code for platforms such as Android or GWT.11

```
val data = try {
    fileContentsToString('data.txt')
  } catch (IOException e) {
    'dummy data'
  }
```

Figure 2.14: Expression example - source11

This is achieved through the XTexd library that runs on top of the Java Development Kit (JDK). The authors state that there is no incompatibility with java, the language is more concise, readable and expressive. It also offers an Eclipse-based IDE integrated with JDT.

[10] http://www.eclipse.org/xtend/
[11] http://www.eclipse.org/xtend/documentation/index.html

2.7.3 TEF - Textual Editing Framework

Textual Editing Framework (TEF) was initially developed by Markus Scheidgen during PhD at the University of Berlin, now is an academic project. [12] TEF provides a syntax definition language o TSL (Textual Syntax Language). TSL describes in textual language (CS) an Ecore (AS) model in the *.etslt* file. For the DSL-to-Model transformation *RunCC* (a parser) is created and interpreted at runtime. The validation of the model is only possible via Eclipse Modeling projects. The figure below shows an example of a CS in the *.etslt* file.(Merkle and Bernhard 2010)

Figure 2.15: TEF Concrete Syntax - source:(Merkle and Bernhard 2010)

Three editors are created through eclipse extension points:

- *Text Editor* - which makes the parse of textual models and allows editing.(Merkle and Bernhard 2010)
- *Model Base Editor* - initially it is a tree editor (other representations are possible) that works as a generic Ecore editor.(Merkle and Bernhard 2010)
- *Embedded Editor* - textual editor built into the tree editor. This editor runs on each element of the template with a hotkey (Alt-T).(Merkle and Bernhard 2010)

The editor supports: syntax highlighting, code completion, navigation and reference, folding and error annotation.(Merkle and Bernhard 2010)

[12] https://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html

2.7.4 MPS - Meta Programming System

Meta Programming System (MPS) created by Jet Brains. The approach of MPS is different because the languages developed are not saved in text form but in an Abstract Syntax Tree (AST) and therefore the concept of *Projection Editor* that allows to edit the AST directly projecting its representation in text. Since there is no text representation of the code it is not necessary a parser which gives a greater flexibility in the construction of the language. The absence of a parser allows to have, in the language, notations of the type decision table or to extend the language for example with mathematical symbols like exponents and matrices. The figure below shows an example. [13]

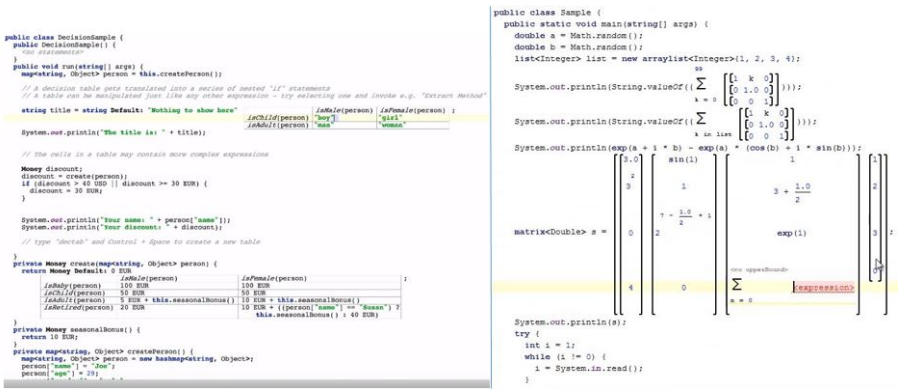


Figure 2.16: MPS Projectional Editor - source:`https://www.jetbrains.com/mps/concepts/`)

The basic notions of MPS are defined below:

- *Node* - form the AST tree. Except for root node, all nodes have child nodes, properties, and references to other nodes. The top elements of the language are the root nodes that are organized into models, for example in Java the root nodes are Classes, Interfaces and Enums. [14]
- *Concept* - defines a "type" of connected nodes, which children, properties, and references an instance of a node can have. In fact, if one concept extends another, it inherits all children, properties, and references from its parent.2.7.4
- *Language* - is a set of *Concepts* with some additional information that includes details in the editors (have to be defined), completion menus, intentions, typesystem and dataflow associated with the language created.2.7.4
- *Generator* - perform a model-to-model conversion from the original model to the target model that is typically defined in another language.2.7.4

2.7.5 EMFText

EMFText is a tool for defining textual syntax for Ecore-based meta-models. The artifacts generated by the EMFText do not contain dependencies and can be customized. The initial

[13] `https://confluence.jetbrains.com/display/MPSD20182/MPS+User's+Guide`
[14] `https://confluence.jetbrains.com/display/MPSD20182/Basic+notions`

syntax for textual DSL can be automatically generated for an Ecore model and complies with the Human-Usable Textual Notation (HUTN) standard. The CS *(Concrete Syntax Specification Language)* is the native language of EMFText based on EBNF allowing a syntax specification to be compact and intuitive. The generated editor contain advanced features like in other code editors such as the Eclipse Java editor, code completion, customizable syntax and occurrence highlighting, advanced bracket handling, folding code, hyperlinks and text hovers, an outline view and instant error reporting . The figure below shows the Ecore model and text definition editor.(EMFText 2012)

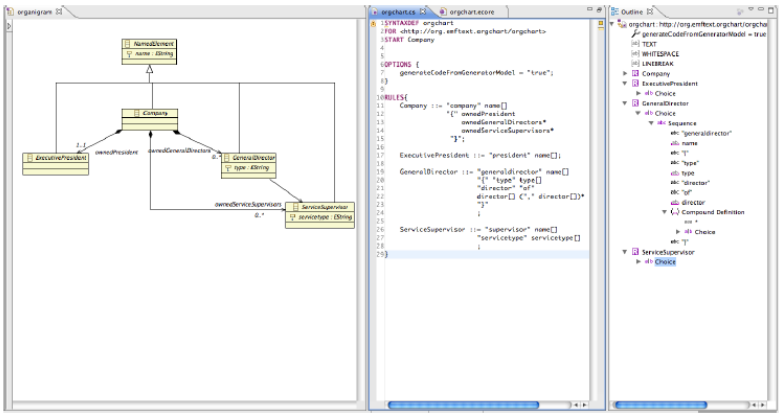


Figure 2.17: EMFText Ecore and Text Definition Editores - source:(Pedro 2009)

The work-flow for creating a new language is an interactive process that consists of the following activities:

- *Specifying a Language Meta-model* - defining the Ecore meta-model (abstract syntax) through the EMF tools that also generate the *.genmodel* file that contains additional information for the code generation, e.g., the path and file information.(EMFText 2012)
- *Specifying the Language's Concrete Syntax* - through the CS *(Concrete Syntax Specification Language)*, a language is defined as textual representation of the concepts of the meta-model (Concrete Syntax). As mentioned above the EMFText is able to automatically produce a specification according to the HUTN from the Ecore meta-model. However it is possible to change the created syntax or start from scratch.(EMFText 2012)
- *Generating the Language Tooling* - EMFText provides two ways to generate the editor and the accompanying infrastructure, either manually in the eclipse (from the context menu) or through an Apache Ant script. EMFText does not have a code generation framework however it is possible to use Eclipse plugins that implement M2M or M2T for example Java Emitter Templates (JET) or Java Merge (JMerge). (EMFText 2012)

2.8 Tool Selection

The choice of the tool for DSL implementation of this work followed the following criteria: learning time, available documentation and tutorials, support and code generation options.

- *Learning Time* - XText and EMFText are similar in terms of concepts and infrastructure created. They are in the context of the tools used in the various master's disciplines and would take the same time to learn, MPS implements the concepts differently with a much longer learning time.
- *Documentation and tutorials available* - XText and MPS have both available good documentation and tutorials, as for the EMFText although there is User Guide and some scattered documents technical information is scarce.
- *Support* - XText and MPS have support through the respective sites, there is a community in both that help to solve possible problems, for EMFText could not find support the site `http://www.emftext.org` is not available.

The TEF is in an experimental phase although it is based on the eclipse platform is far from the other tools so it is not considered in the comparison.

The choice of the author is XText, because it is within the context of the tools used in the various master's disciplines and therefore with a short learning curve, there is support and tutorials, and an active community.

Chapter 3

Value Analysis

This chapter describes the value analysis according to Peter Koen's New Concept Development Model (NCD). The value of a product is interpreted differently by different people. Some of the characteristics that influence this evaluation are performance, capacity, emotional value and style, in relation to its cost. Lawrence Miles created the value analysis in 1945. Later Peter Koen (Koen et al. 2001) developed a set of processes with the objective of innovation for the development of new products and improvement of existing products. This thesis uses the NCD.

3.1 New Concept Development Model (NCD)

A group of companies wanted to compare best practices in the Fuzzy Front End (FFE) innovation process. Because there was neither a common language nor a definition of the key elements of the front end process the process of comparison became unfeasible. The solution was to develop a set of processes, represented with NCD model. This model is divided into three key parts: five front end elements, the engine that powers the elements, and external influencing factors (Koen et al. 2001). Now, with a common language for defined key activities and concepts, the FFE presents the greatest number of opportunities in the innovation process. This phase defines the activities that are carried out before the formal and well structured of the New Product and Process Development (NPPD).

The engine represents the support of the company's executives, leaders, and organizational culture, which provides the power needed by the five elements of the NCD model.The external part takes into account the external factors that influence the decisions of the internal elements. The two inward-pointing arrows (for Idea Genesis and Opportunity Identification elements) act as starting points for innovation. The arrow pointing out from the Concept & Technology Development element, which acts as the last element of the model, implies the development of a business plan based on estimates for a potential market and customer needs (Koen et al. 2001).

This thesis is based on the opportunity provided by the joint adoption of several technologies: MDD, REA and DSL to build an text DSL of REA ontology, something that by the analysis of the literature has not yet been done.

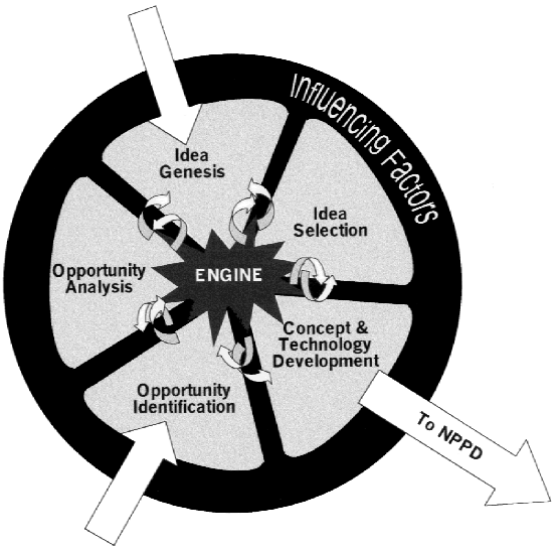


Figure 3.1: The New Concept Development Model (NCD) provides a common language and definition of the key components of the Front End of Innovation.

3.2 Front-End Elements

- *Opportunity Identification*

  Organizations identify the opportunities they wish to promote. Opportunities and business technologies are considered here to allocate resources in emerging areas and emerging markets. The direction is the one that the organization has as business objective. For example, the opportunity may be a near-term response to a competitive threat, a breakthrough possibility for capturing competitive advantage, or a means to simplify / speed-up / reduce the cost of operations. The opportunity could be an entirely new direction for the business or a minor upgrade to an existing product. It could also be a new product platform, a new manufacturing process, a new service offering, or a new marketing or sales approach. Creativity tools and techniques (e.g. brainstorming, mind mapping and lateral thinking) as well as problem-solving techniques (e.g., causal analysis, fish-bone diagrams, process mapping, and constraint theory) can be used (Koen et al. 2001).

- *Opportunity Analysis*

  Identifying opportunities requires additional information so it can be translated into business opportunities and technologies. Making assessments at this early stage is taking risks because almost everything is uncertain. The effort expended must adapt to the attractiveness of the opportunity, the effort in the future development, the business strategy, culture and risk tolerance of each organization. This element may be part of a formal process or may occur interactively as a reaction to the opportunities presented. "what-if" scenarios may be useful in this element (Koen et al. 2001).

  The company Numbersbelieve is developing the OMNIA platform and need the formalization of the REA ontology into an textual DSL. The DSL of this work can be tested in the business world on a real application .

Because REA specifies the fundamental laws of the business domain, knowledge of these laws increases the potential of application designers to design applications without omissions which guarantees them consistency from a business perspective. Designing REA-based applications is concise and easy to understand for users of these applications, consultants, and application developers Hruby 2013.

- *Idea Genesis*

  Creating, developing, and maturing an opportunity into a concrete idea represents an evolving process in which ideas are created, rejected, combined, remodeled, modified, and updated. Direct contact with clients / users, with multidisciplinary teams and with other organizations increases the likelihood that an idea, which has undergone several iterations, has been subject to change, studied and discussed, to be chosen / selected. However a new idea can come from a more formal process including brainstorming sessions and a set of ideas. From outside the formal process may also come up with new ideas - a supplier can provide a new material or a user can make an unusual request. As the elements of the NCD may or may not be processed in sequence, ideas produced here may feed other elements of the process. Typically this element produces a fully developed idea or product concept (Koen et al. 2001).

  Several possibilities were put forward for discussion using some brainstorming sessions but also a set of ideas were used based on methodologies that implemented the concepts need: MDD, REA and DSL

- *Idea Selection*

  In many organizations there are so many product/process ideas that the critical activity is to choose which ideas can be transformed into value for the business. The selection process may be simple as an individual or a more formal choice according to a predefined method. At this stage Front End of Innovation (FEI) no definition of return value is possible. For this reason it is necessary to have a selection model that takes into account factors such as technological risks, investment levels, competitive realities, organizational capacity (Koen et al. 2001).

  The selection was made according to the objectives of this thesis, verify if a certain set of recent approaches (REA, DSL can be used to formalize an textual DSL that can be used with the OMNIA platform.

- *Concept and Technology Development*

  The final step is processed in this element of the model. It is necessary to develop a business model case based on estimates of a potential market, needs of the clients, needs of investments, analysis of the competition, choice of technologies and analysis of risk of the project. The level of formality varies according to the nature of the idea / opportunity (Koen et al. 2001).

  The Business Model Canvas was developed. The technologies chosen was Eclipse as an IDE, EMF as a modeling tool and XText to develop the text DSL. Based on these technologies the following artifacts will be produced: a meta-model (abstract syntax) and a textual DSL (concrete syntax) based on the REA ontology.

3.3 Value, Perceived value and Customer value

3.3.1 Value

The value of a particular product reflects the desire of the owner to sell or of the purchaser to obtain the product, this introduces subjective aspects to the value of a product. Value engineering considers the cost in relation to the function and recognizes that there is relation of three dimensions between, function, cost, and value (Dell'Isola, Alphonse J. 1982). So the value of a product includes the cost and a marginal value that depends on which is the value system of the owner and the buyer. This marginal value is the subjective part and reflects the desire to obtain (buyer) or maintain (owner) the product or how much they are willing to pay for reasons of prestige, appearance, aesthetic, judicial, religious or moral, or any combination of reasons. Only if the subjective part can be evaluated in monetary terms can the value of a product be expressed in monetary units (Neap, H.S. & Celik, T. International Journal of Value-Based Management 1999).

3.3.2 Perceived value

In Cambridge Business English Dictionary is a method of pricing a product based on how much customers are willing to pay for it rather than on how much it costs to produce. A customer's opinion about the value of a product may have little or nothing to do with the market price of the product and depends on the ability of the product to meet its needs or requirements [1])

3.3.3 Customer value

"Customer value is the satisfaction the customer experiences (or expects to experience) by taking a given action relative to the cost of that action. The given action is traditionally a purchase, but could be a sign-up, a vote or a visit, while the cost refers to anything a customer must forfeit in order to receive the desired benefit, such as money, data, time, knowledge" [2]. It is possible to develop models and equations to determine value for the customer. A simple equation can be:

Perceived Value = Perceived Benefits / Cost

For a set of benefits, as the cost increases, the perceived value goes down. The value does not refer to the price. It refers to the benefits obtained in the context of the price [3].

The following diagram shows the benefits and sacrifices for the client.

3.4 Value Proposition

There are many organizations where the method of development continues to be traditional methods of programming languages, frameworks and middleware. Customers point out

[1] http://www.businessdictionary.com/definition/perceived-value.html
[2] https://builtvisible.com/understanding-customer-value/)
[3] https://builtvisible.com/understanding-customer-value/)

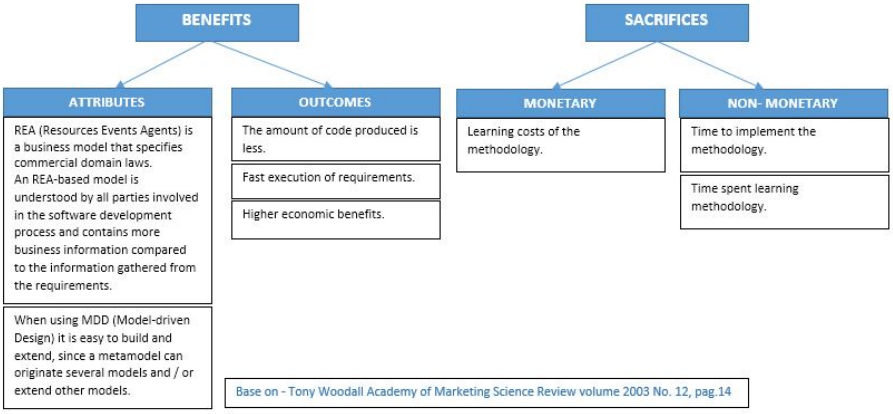


Figure 3.2: Benefits and Sacrifices for the client

several difficulties: lack of flexibility, requirements delivered out of time, requirements out of budget, lack of flexibility and updating apps takes to long (Rymer 2017).

Using MDD approach it is possible to focus on a context, the REA ontology, ignoring development environments and platforms. The meta-model and textual DSL created, due to the level of abstraction, are more adaptable to the OMNIA platform than conventional code. The result are artifacts that can be tested and altered by everyone.

3.5 CANVAS Business model

As stated by Osterwalder the CANVAS model is a tool that must be used for the strategic management of organizations (Osterwalder and Pigneur 2011).

The nine fundamental groups, described in the CANVAS model (Osterwalder and Pigneur 2011) , for the business model in this thesis:

- *Customer Segments* - An organization serves one or several customer segments And a CANVAS must be made for each segment. Companies and individuals who wish to use the MDD, REA ontology and DSL approaches for the design of information systems.
- *Value Propositions* - It seeks to solve customer problems and satisfy their needs with.

  Create applications with the use of declarative tools, model-driven development and less technical knowledge are the value propositions for this segment of customers.
- *Channels* - Communication, distribution and sales channels are the means to reach customers.

  The GIT repository is the channel to use.
- *Customer Relationships* - Different customer segments can have different types of relationships.

  Customer relationship defines Error Correction and Repository Sharing.
- *Key Activities* - Key activities for the implementation of value propositions.

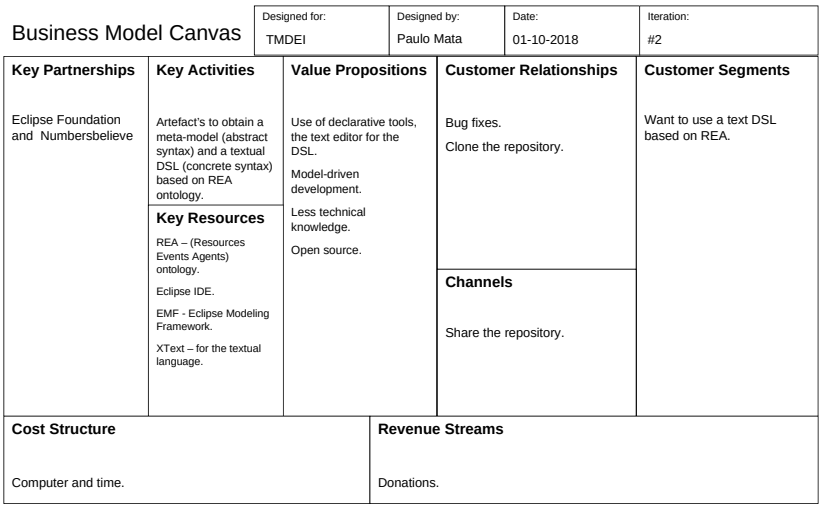


Figure 3.3: CANVAS Business model

Construct the necessary artefact's to obtain a meta-model (abstract syntax) and a textual DSL (concrete syntax) based on REA ontology.

- *Key Resources* - They are the active resources necessary to be able to deliver value propositions.

  The tools necessary to implement the value prepositions, REA, Eclipse IDE, XText, EMF and related technologies.

- *Key Partnerships* - Some activities are outsourced some resources may be procured outside the organization. Suppliers and sub-contractors are the Partners.

  For this work the Numbersbelieve and Eclipse Foundation Foundation are the key partners.

- *Cost Structure* - Describes costs to provide value propositions.

  This work requires a Computer and time.

- *Revenue Streams* - Define how to get revenues from the value propositions.

  The revenues for this work are donations.

Chapter 4

Analysis and Design

This chapter shows the process of analysis, highlighting both functional and non-functional requirements.

It describes the process of analyzing the REA requirements based on the essential economic processes present in any organization. Chapter Two already describes the fundamental concepts of REA. From the image2.4 in chapter two it is possible to identify: *Economic Resource, Economic Agent, Economic Event, Commitment, and Contract*. These concepts are analyzed in the context of an organization in the following sections.

The DSL design uses the activities proposed in the *"An approach to the systematic development of domain-specific languages"* as guidelines. The proposed activities are the result of the author's experience in the development of DSL projects in an Model Driven Software Development (MDSD) context.(Strembeck and Zdun 2009)

4.1 Organization Trading Partners

An organization can increase or decrease the value of its resources typically by exchanging those resources. Exchange is the process in which an organization receives economic resources from another economic agent, and provides resources to this economic agent in return. (Hruby 2013) The image below shows the economic agents essential to the functioning of an organization.

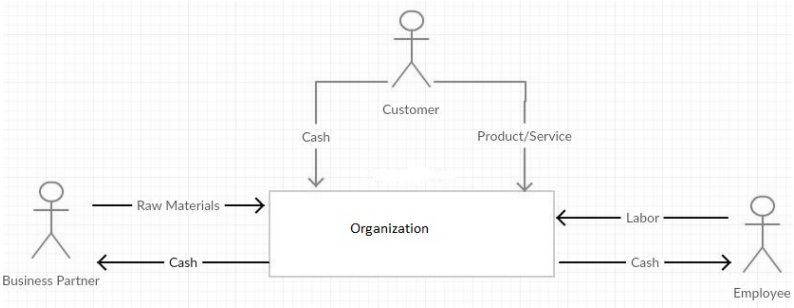


Figure 4.1: Organization trading partners - Base on:(Hruby, P. 2013, p.7)

The essential trading partners of an organization are: it's customers, the employees, and who sells the raw materials, the sellers. Each of them defines an economic agent.(Hruby 2013)

4.2 Organization Trading Processes

The main business processes of an organization are: sales of products or services *(the Sales process)*, purchase of raw materials from sellers *(the Purchase process)*, and labor acquisition *(the Labor Acquisition process)*. The image below shows these processes as the REA version of the Porter's value chain. REA provides an accurate definition of what a business process is *(a set of economic events related by the duality relationship)*.(Hruby 2013)

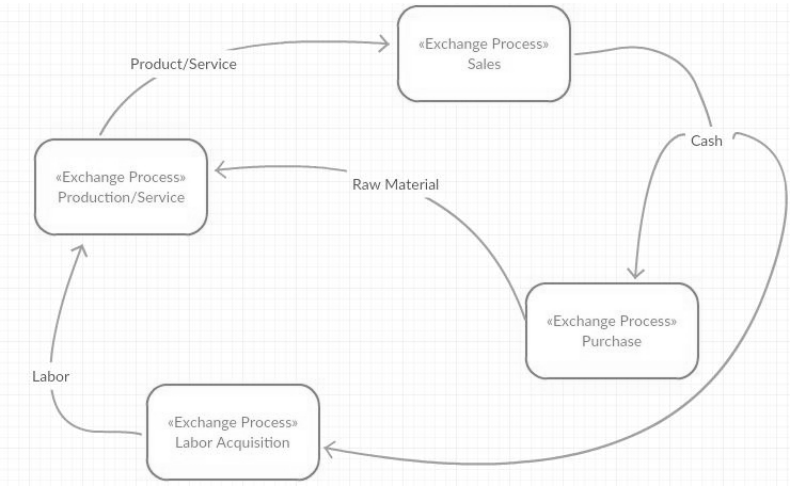


Figure 4.2: Organization - REA value chain - Base on:(Hruby, P. 2013, p.8)

In the following sections the REA models are defined for each process.

4.2.1 Sales Process

Selling products or services to customers is an important economic event for organizations. This economic event can be defined as an exchange of goods or services for money. The *Sale Process* represents an outflow of products or services and an inflow of money.(Hruby 2013) The REA model of the *Sales process* focuses on the core economic phenomenon,

Figure 4.3: Use Case Sales Process - Base on:(Hruby, P. 2013)

because of this it covers several more specific cases, for example customers can pay at the time of purchase or use of the service, others receive an invoice and pay after a period of time, customers can pay with cash, card or by check. These cases and others are covered by the model.(Hruby 2013)

The *Organization* and the *Customer* are economic agents, the *Product or Service* are economic resources. Transferring ownership of the Product/Service from the organization to the Customer *(the Sale)* is an economic event. Reciprocally there is an economic event that is the transfer of ownership of Cash from the customer to the organization *(the Cash*

Receipt). For the organization the *Sale event* is a decrement event because it decreases the value of the organization's resources. The *Cash Receipt* is an increment event because it increases the value of the resources under control of the organization. *Sales Order* is an example of a contract between the organization and the customer. *Sale Lines* and *Payment Line* are not economic events, they are commitments to execute economic events in the future. *Sales Line* is a commitment to perform the *Sale event* while *Payment Line* is a commitment to the *Cash Receipt* event in the future. (Hruby 2013) The model described is defined in the image below.

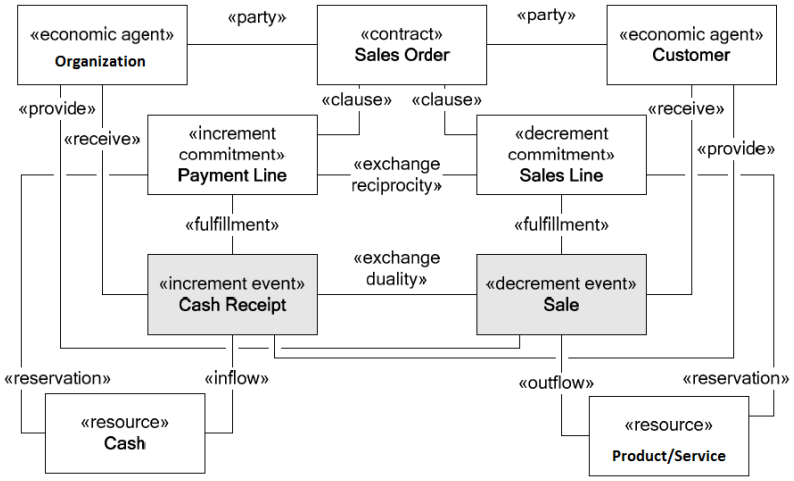


Figure 4.4: REA Model - Sales Process With Contract Based on:(Hruby, P. 2013)

4.2.2 Purchase Process

An organization needs raw materials that transforms into new products or subcontracts services. This economic event can be defined as an exchange of money for goods or services. The *Purchase process* represents an outflow of money and an inflow of raw materials or services to the organization.(Hruby 2013) The REA model of the *purchase process* cov-

Figure 4.5: Use Case Purchase Process - Based on:(Hruby, P. 2013)

ers several specific cases. The raw material can be paid by check, bank transfer and even in different currencies. The model tracks which *Purchase* matches which *Cash Disbursements*.(Hruby 2013)

The *Vendor* and the *Organization* are economic agents, *Cash* and *Raw Material* are economic resources. The transfer of ownership of *Raw Material* to the organization is an increment event *(the Purchase)*, the transfer of ownership from *Cash* to *Vendor (Cash Disbursement)* is a decrement event.(Hruby 2013) The model described is defined in the image below.

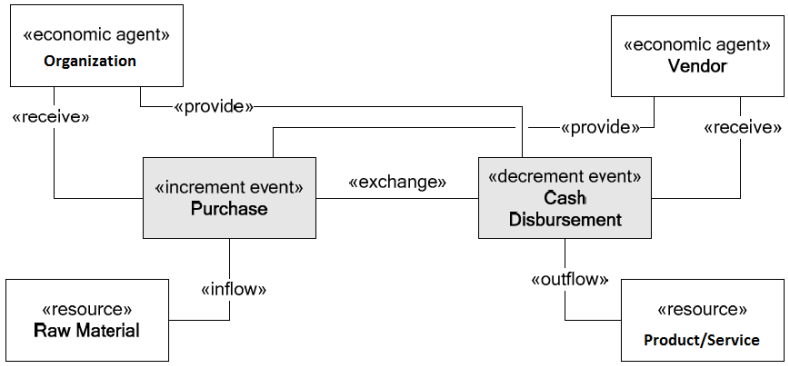


Figure 4.6: REA Model - Purchase Process - Based on:(Hruby, P. 2013)

4.2.3 Labor Acquisition Process

The organization's employees provide work and receive a salary in return. This economic event can be defined as the exchange of labor for money. The *Labor Acquisition* process represents an outflow of *Cash* and an inflow of *Labor*.(Hruby 2013) The employee and the

Figure 4.7: Use Case Labor Process - Based on:(Hruby, P. 2013)

organization are economic agents. *Labor* and *Cash* are economic agents. *Labor Acquisition* occurs for time periods (worked hours), *Cash (Cash Disbursement)* occurs once a week or a month. The REA model can be applied to various forms of work: full employment, temporary work, consulting and other formats.(Hruby 2013) The model described is defined in the image below.

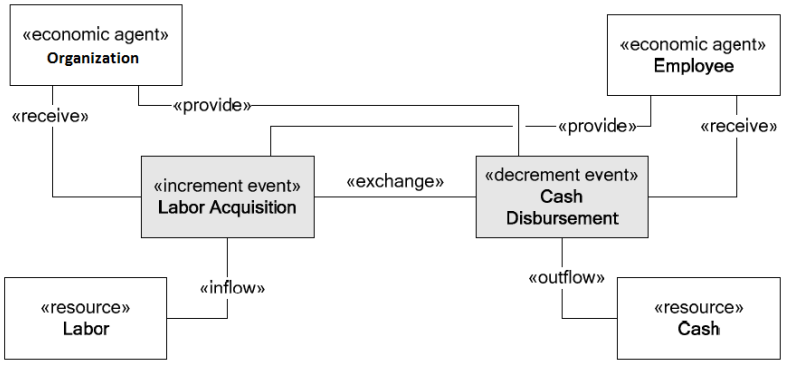


Figure 4.8: REA Model - Labor Acquisition Process - Based on:(Hruby, P. 2013)

4.3 Non-functional Requirements

These are the non-functional requirements according to the following classifications:

- Usability:
  - Technical and non-technical should be able to use DSL.
  - DSL must be self-explanatory and intuitive.
  - When using or changing the language and after the user saves the changes the editor should automatically create a Json file according to the specifications.
  - The IDE (editor) should enable syntax and semantic error detection, and context assistance with auto-complete.
- Flexibility: The editor should be able to run on different systems, for example Windows or Linux.

4.4 Language Model Driven

Language model driven is one of the engineering processes for the development of DSLs. As the root of the process is the definition of a language model that contains all abstractions of the domain (the abstract syntax), the next step is the definition of the concrete syntax while defining the behavior (this will not be define) of the DSL and finally the mapping of the DSL to the platform on which DSL runs. The image below shows the table with the proposed activities.(Strembeck and Zdun 2009)

Table I. Overview: main activities in our DSL development approach.

Main activity	Sub-activities	Typical input	Typical output
Defining the DSL's core language model	Identify domain abstraction, add domain abstraction to language model, define language model constraints, check language model	Depends on the actual process tailoring	DSL core language model, DSL language model constraints
Defining the DSL's concrete syntax(es)	Define symbols for language model elements, define DSL production/composition rules, define DSL concrete syntax	DSL language model	DSL concrete syntax
Integrating DSL artifacts with the platform/infrastructure	Map DSL artifacts to platform features	DSL language model, DSL concrete syntax	Transformations

Figure 4.9: Main activities in DSL development approach - source:(Strembeck and Zdun, 2009)

There are other processes such as *Mockup language* (which promotes the participation of domain experts) and *Extract DSL from existing systems*, however the process followed for design in this work is *Language model driven*.(Strembeck and Zdun 2009)

4.5 Define DSL Core Language Model

As stated above, the DSL core language model (abstract syntax) defines all abstractions of the domain, for this dissertation the domain is the REA ontology. In this activity, a REA core language model and its constraints are defined.

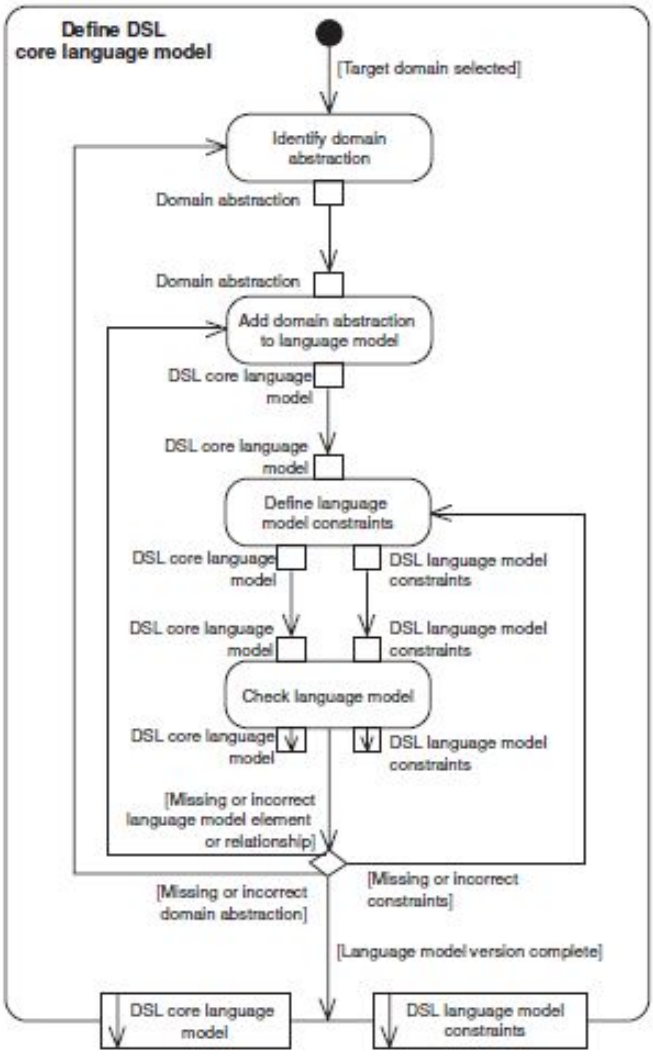


Figure 4.10: Subprocess- Define DSL Core Language Model- source:(Strembeck and Zdun, 2009)

The REA abstractions for the DSL definition are:

- Resource.
- Event - Increment Event and decrement event.
- Agent - Internal Agent and External Agent.
- Contract.
- Commitment- Increment Commitment and Decrement Commitment.

Some of the constraints that derive from the axioms of REA ontology, already mentioned, they are:

- For each Resource there must be a "take" and a "give" event.
- All events affecting the decrease of a resource must be linked in a "duality" relationship with incremental events.
- Each economic event must be related to an economic agent ("inside" and "outside") through a "provide" and a "receive" relationship.

4.6 Define DSL Concrete Syntax

The concrete syntax of a DSL can be defined in graphic or text form. In the context of this dissertation, the syntax of the REA DSL is textual. Implementation of the DSL is achieved through XText editor.

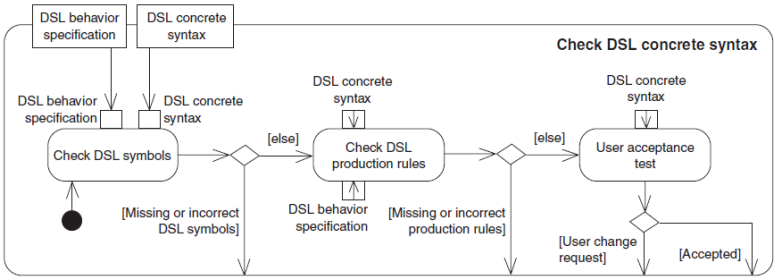


Figure 4.11: Subprocess- Checking Concrete Syntax Artifacts - source:(Strembeck and Zdun, 2009)

The abstract syntax symbols will be defined in concrete syntax according to the following production rules:

- *Resource* defined with:
  - name = ID
- *Event* - Decrement and Increment.
- *Increment Event* defined with:
  - name = ID
  - take = Increment Event
  - give = Decrement Event
  - duality = Decrement Event
  - resource = Resource
  - receive = Internal Agent
  - fulfillment = Increment commitment

  - participants = List of Agents
- *Decrement Event* defined with:
  - name = ID
  - take = Increment Event
  - give = Decrement Event
  - duality = Increment Event
  - resource = Resource
  - receive = External Agent
  - fulfillment = Decrement commitment
  - participants = List of Agents
- *Agent* - Internal and External.
- *Internal Agent* defined with:
  - name = ID
  - provide = Increment Event
  - external participant = External Agent
  - internal participant = Internal Agent
- *External Agent* defined with:
  - name = ID
  - provide = Decrement Event
  - external participant = External Agent
  - internal participant = Internal Agent
- *contract* defined with:
  - name = ID
  - internal party = Internal Agent
  - external party = External Agent
  - increment clause = Increment Commitment
  - decrement clause = Decrement Commitment
  - commitments = List of Commitment
- *Commitment* - Increment and Decrement.
- Increment Commitment defined with:
  - name = ID
  - reciprocity = Decrement Commitment
  - reservation = Resource

- Decrement Commitment defined with:
  - name = ID
  - reciprocity = Increment Commitment
  - reservation = Resource

4.7 Mapping of the DSL to the Platform

In this activity, DSL transformation templates are defined for the OMNIA platform. The transformation generates a Json file that conforms to the specifications provided by Numbersbelieve to the OMNIA platform. Data from a DSL instance is used through the OMNIA API to create an application on the platform.

"These transformations should not rely directly on the concrete syntax. That is, if more than one concrete syntax is defined, the same (generic) transformations should work for all concrete syntaxes."(Strembeck and Zdun 2009)

The mapped concepts are:

- Resources.
- Events.
- Commitments.

Chapter 5

Implementation

This chapter describes the entire development process. The development of the meta-model (abstract syntax), the language (concrete syntax), editor for the language, and the Json file. The code editor used in development process is the Obeo Designer Community (Version 10.1.2) [1] that already has integrated the tool XText and EMF tools. Other versions of Eclipse can be used such as *Eclipse Modeling Tools* [2]. Three languages and a meta-model are described identified by the name of the project, "REA_DSL_TMDEI" (the meta-model), "tmdei.rea.reav3", "tmdei.rea.reav4" and "tmdei.rea.dsl4all".

5.1 Installation and Project

To install the *Obeo Designer Community* download the correct version and unzip the Zip file to an appropriate location. To open the program run the *obeodesigner.exe* file and make the necessary updates. To create a new EMF project through the editor, go to *File » New » Project » Other* menu and select *Eclipse Modeling Framework » Ecore Modeling Project*. This starts a new project wizard, after the necessary information is set (e.g. project name) a file structure is created. The image below shows the wizard and the project file structure.

5.2 Meta-Model

The important files for the meta-model are *.ecore* and *.genmodel*. The *.ecore* contains information about the created classes that describe the model and the *.genmodel* contains information about the generation of code [3] .

5.2.1 Ecore

In the ecore file it is possible to define the following elements:

- *EClass* - represents a class, with zero or more attributes and zero or more references5.2.

[1] `https://www.obeodesigner.com/en/download`
[2] urlhttps://www.eclipse.org/downloads/packages/
[3] `http://www.eclipse.org/modeling/emf/docs/`

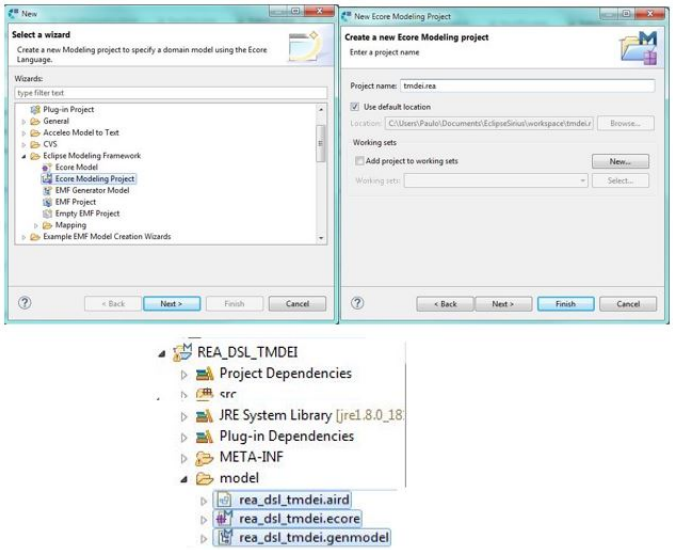


Figure 5.1: Create a New Modeling Project

- *EAttribute* - represents an attribute which has a name and a type5.2.
- *EReference* - represents one end of an association between two classes. It has flags to indicate if it represents a containment and a reference class to which it points5.2.
- *EDataType* - represents the type of an attribute, e.g. *java.util.Date, int, or float*5.2.

For the definition of the REA meta-model, two *.ecore* files were created, first defining the *Economic Event* concepts, and a second one extends the first with the definition of REA *Contract and Commitment* concepts. The images below show these concepts in a tree format (the Ecore format), the association and cardinality between them, and the attributes.

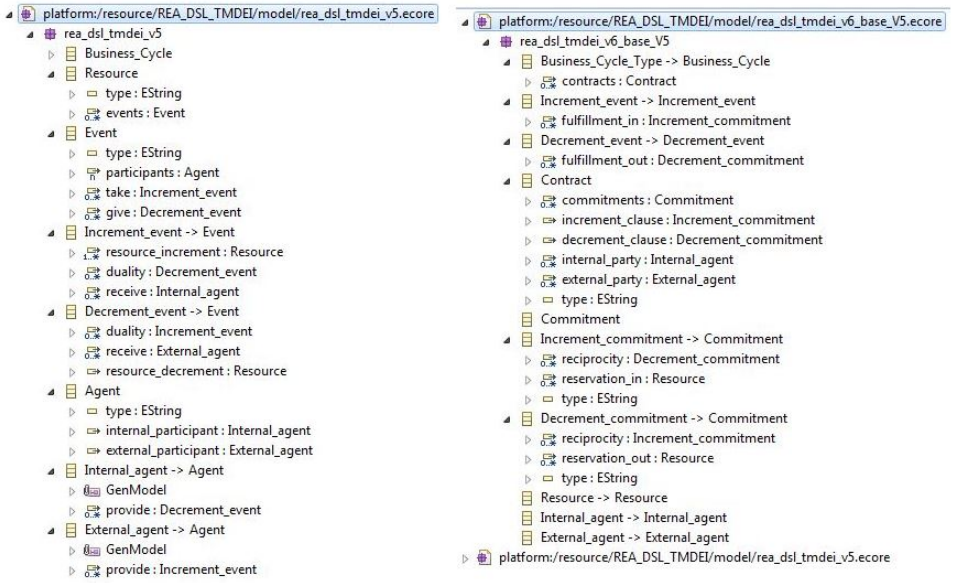


Figure 5.2: Ecore - Economic Event, Contact and Commitment concepts

5.2.2 Ecore Graphic Representation

Once the *ecore* file is defined it is possible to obtain a graphic representation of the model, through the context menu and selecting *Initialize Ecore Diagram*. However if the *.ecore* file is empty the previous method shows an empty workbench. The graphical tools can be used to define a model and this definition is automatically reflected in the *.ecore* file, and vice-versa. The images below show the graphical representation of the two *.ecore* files mentioned above.

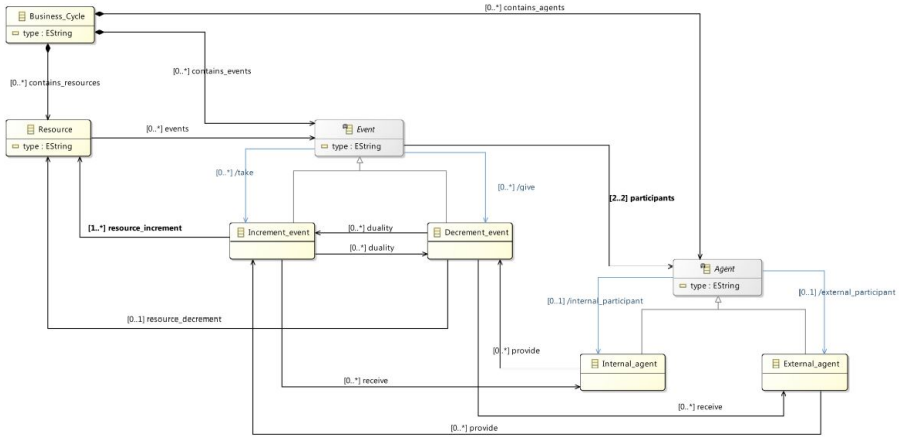


Figure 5.3: Graphic Representation Ecore - Economic Event concepts

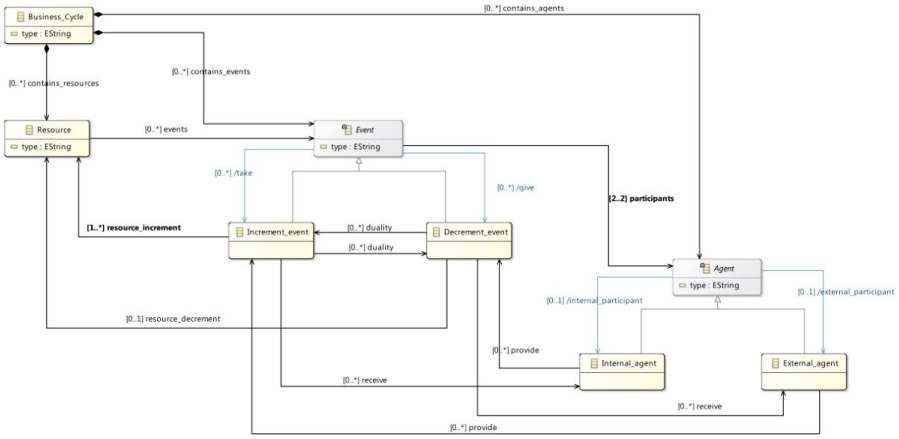


Figure 5.4: Graphic representation Ecore - Contract and Commitment concepts

5.3 DSL with XText

This section describes the activities required to define the DSL with XText.

5.3.1 XText Project

The first step is to create the XText project as described below:

- *File » New » Project » select: Xtext, choose: Xtext Project and click Next*
- *Project name: tmdei.rea.reav3*
- *(Language) Name: tmdei.rea.reav3.Reav3Dsl* (the last word should start with a capital letter following Java conventions)
- *(Language) Extension: reav3dsl*
- *Finish*

XText automatically creates an infrastructure of five projects:

- *tmdei.rea.reav3*: Contains the grammar definition and all related components such as parser, lexer, linker, validation, etc.
- *tmdei.rea.reav3.ide*: Platform-independent IDE functionality.
- *tmdei.rea.reav3.ui*: Contains the DSL text editor and other workbench related functionality. In this project, advanced functionality can be implemented, such as content assist, outline tree, and quick fix functionality.
- *tmdei.rea.reav3.tests*: The place for unit tests for the DSL.
- *tmdei.rea.reav3.ui.test*: Unit tests for the Eclipse editor.

There are three files relevant to the DSL development process. Two are located in the *tmdei.rea.reav3* project in the *src* folder and on package *tmdei.rea.reav3 (GenerateReav3Dsl.mwe2 and Reav3Dsl.xtext files)* and the third in the *src* folder package *tmdei.rea.reav3.generator (Reav3DslGenerator.xtend file)*. Below is a brief description of each:

- *GenerateReav3Dsl.mwe2* - *.MWE2* (Modeling Workflow Engine) allows you to compose object graphs declaratively. Defines a wokflow of components that interact among them for example components to read, perform operations (transformations) on EMF resources, and information to generate other artifacts examples of that information are: project and language name and the extension of language files. [4]
- *Reav3Dsl.xtext* - define the grammar. When first opened in the editor it shows an example of grammar.
- *Reav3DslGenerator.xtend* - for code generation and model transformation.

5.4 DSL's Definition

This section shows the definition of the DSL's syntax and transformations (e.g. code generation). Placing the definition of DSL's syntax in a MOF layer will be the M2 layer *Metamodels*. The image below shows two layers MOF M2 (meta-model) and M1 (model) that divide the development process of the DSL.[5]

[4] `https://www.eclipse.org/Xtext/documentation/306_mwe2.html`
[5] Xtext_DSL_GettingStarted_Neon.pdf

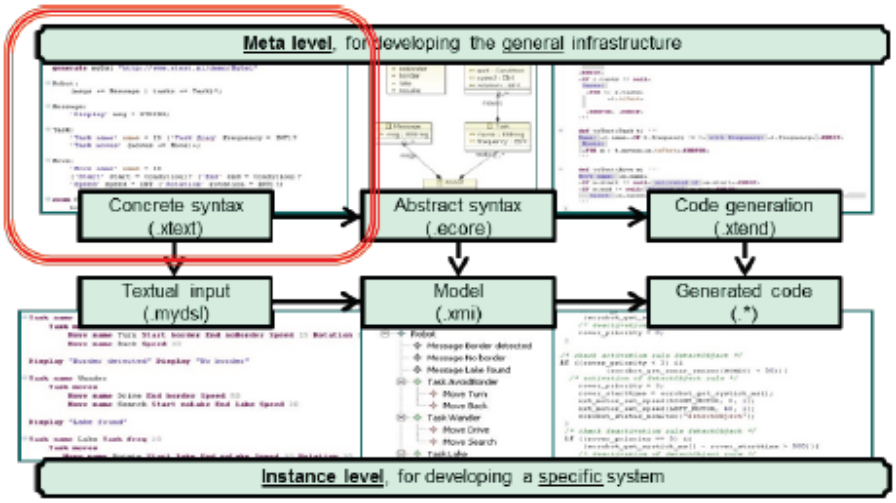


Figure 5.5: Meta-model level source:5.4

5.4.1 Common Terminals

Xtext comes with a default set of predefined and often required terminal rules. The grammar for these common terminal rules is defined as follows: [6]

```
grammar org.eclipse.xtext.common.Terminals
    hidden(WS, ML_COMMENT, SL_COMMENT)

import "http://www.eclipse.org/emf/2002/Ecore" as ecore

terminal ID:
    '^'?('a'..'z'|'A'..'Z'|'_')('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;

terminal INT returns ecore::EInt:
    ('0'..'9')+;

terminal STRING:
    '"' ( '\\'('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|'"') )* '"' |
    "'" ( '\\'('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|"'") )* "'";

terminal ML_COMMENT:
    '/*' -> '*/';

terminal SL_COMMENT:
    '//' !('\n'|'\r')* ('\r'? '\n')?;

terminal WS:
    (' '|'\t'|'\r'|'\n')+;

terminal ANY_OTHER:
    .;
```

Figure 5.6: XText Common Terminals - source:5.4.1

[6] http://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html#common-terminals

5.5 Building the DSL Grammar

This section shows the construction activities of the grammar *(tmdei.rea.reav3)* describing the production rules (or actions) and the terminal and non-terminal symbols. The first line declares the name of the language and the clause ("with") declares that this grammar is extended by the grammar which contains the terminal symbols of XText. On the second line the *generate* declaration in the grammar advises the XText framework to infer the Ecore model from the grammar. [7]

5.5.1 Resource - Production rule

The *Resource* production rule starts with an identifier (or feature) *name* which is parsed by a rule called *ID* (defined in the super grammar). The value returned by the call to *ID* is assigned (=) to the feature *name*.

5.5.2 Event - Production rule

The *Event* production rule is defined as the choice ("|") between two types of actions (Increment_Event and Decrement_Event). The choice is modeled in the Ecore model by the relation *SuperType*. Events are related to economic events.

5.5.3 Increment_Event - Production rule

The *Increment_Event* action define several keywords and features:

- The rule starts to define two keywords *'Increment' and 'Name'* and a feature *name* which is assigned the value returned by rule *ID*. After each feature the non-terminal symbol *','* appear.
- The *'Company_Take'* keyword and the feature *take* are defined. The feature *take* is a cross reference, denoted by square brackets use, to other rule. The square brackets in the text grammar are modeled as a relation to an EClasse (in this case to Increment_Event) Object in the Ecore model. This feature it's *Increment_Event* type.
- The *'Company_Give'* keyword and the feature *give* are defined. It is also a cross reference. This feature it's *Decrement_Event* type.
- *'Duality_Relation'* keyword and the feature *duality* are defined. It is also a cross reference. This feature it's *Decrement_Event* type. It represents the relation to a decrement event.
- The *'Company_Resource_Increment'* keyword and the feature *resource_increment* are defined. It is also a cross reference. This feature it's *Resource* type and represents the resource that will be incremented.
- The *'Internally_Received_By'* keyword and the feature *receive* are defined. It is also a cross reference. This feature it's *Internal_Agent* type, it represents the internal agent related to this event.

[7] https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html

- The *'Fulfillment_in'* keyword and the feature *fulfillment_in* are defined. It is also a cross reference. This feature it's *Increment_commitment* type. It represents a list of elements indicated by the symbols "+=", the list can have zero or more elements indicated by the symbol "*" and this feature is optional indicated by the symbol "?". The elements of the list appear in parentheses and separated by white spaces.
- The *'Event_Participants'* keyword and the feature *participants* are defined. It is also a cross reference. The feature it's *Agent* type. It's a list of agents that participate in an economic event and can be related to this event. Elements in the list are separated by commas.

5.5.4 Decrement_Event - Production rule

The *Decrement_Event* action define several keywords and features:

- The rule starts to define two keywords *'Decrement' and 'Name'* and a feature *name* which is assigned the value returned by rule *ID*. After each feature the non-terminal symbol *','* appear.
- The *'Company_Take'* keyword and the feature *take* are defined. The feature *take* is a cross reference. This feature it's *Increment_Event* type.
- The *'Company_Give'* keyword and the feature *give* are defined. It is also a cross reference. This feature it's *Decrement_Event* type.
- *'Duality_Relation'* keyword and the feature *duality* are defined. It is also a cross reference. This feature it's *Increment_Event* type. It represents the relation to a increment event.
- The *'Company_Resource_Decrement'* keyword and the feature *resource_decrement* are defined. It is also a cross reference. This feature it's *Resource* type and represents the resource that will be decremented.
- The *Externally_Received_By'* keyword and the feature *receive* are defined. It is also a cross reference. This feature it's *External_Agent* type, it represents the external agent related to this event.
- The *'Fulfillment_out'* keyword and the feature *fulfillment_out* are defined. It is also a cross reference. This feature it's *Decrement_commitment* type. It represents a list of elements indicated by the symbols "+=", the list can have zero or more elements indicated by the symbol "*" and this feature is optional indicated by the symbol "?". The elements of the list appear in parentheses and separated by white spaces.
- The *'Event_Participants'* keyword and the feature *participants* are defined. It is also a cross reference. The feature it's *Agent* type. It's a list of agents that participate in an economic event and can be related to this event. Elements in the list are separated by commas.

The image below shows the Events definition.

```
Event:Increment_Event | Decrement_Event

Decrement_Event: 'Decrement'
            'Name' name=ID ','
            'Company_Take' take=[Increment_Event] ','
            'Company_Give' give=[Decrement_Event] ','
            'Duality_Relation' duality=[Increment_Event] ','
            'Company_Resource_Decrement' resource_decrement=[Resource] ','
            'Externally_Received_By' receive=[External_Agent] ','
            ('Fulfillment_out' '('fulfillment_out+=[Decrement_commitment]*')'',')?
            'Event_Participants' '('participants+=[Agent] ',' participants+=[Agent] ')' ';'

Increment_Event: 'Increment'
            'Name' name=ID ','
            'Company_Take' take=[Increment_Event] ','
            'Company_Give' give=[Decrement_Event] ','
            'Duality_Relation' duality=[Decrement_Event] ','
            'Company_Resource_Increment' resource_increment=[Resource] ','
            'Internally_Received_By' receive=[Internal_Agent] ','
            ('Fulfillment_in' '('fulfillment_in+=[Increment_commitment]*')'',')?
            'Event_Participants' '('participants+=[Agent] ',' participants+=[Agent] ')' ';'
```

Figure 5.7: Events

5.5.5 Agent - Production rule

The *Agent* action rule is defined as the choice ("|") between two types of actions (Internal_Agent and External_Agent). The choice is modeled in the Ecore model by the relation *SuperType*.

5.5.6 External_Agent - Production rule

The *External_Agent* action define several keywords and features:

- The rule starts to define two keywords *'External_Agent' and 'Name'* and a feature *name* which is assigned the value returned by rule *ID*. After each feature the non-terminal symbol *','* appear.
- The *'That_Provide'* keyword and the feature *provide* are defined. The feature *provide* is a cross reference. This feature it's *Increment_Event* type.
- The *'By_External_Participant'* keyword and the feature *external_participant* are defined. It is a cross reference. This feature it's *External_Agent* type. It relates to *this* agent.
- The *'To_Internal_Participant'* keyword and the feature *internal_participant* are defined. It is a cross reference. This feature it's *Internal_Agent* type. It relates the internal agent to this agent.

5.5.7 Internal_Agent - Production rule

The *Internal_Agent* action define several keywords and features:

- The rule starts to define two keywords *'Internal_Agent' and 'Name'* and a feature *name* which is assigned the value returned by rule *ID*. After each feature the non-terminal symbol *','* appear.

- The *'That_Provide'* keyword and the feature *provide* are defined. It is a cross reference. This feature it's *Decrement_Event* type.
- The *'To_External_Participant'* keyword and the feature *external_participant* are defined. It is a cross reference. This feature it's *External_Agent* type. It relates the external agent to this agent.
- The *'By_Internal_Participant'* keyword and the feature *internal_participant* are defined. It is a cross reference. This feature it's *Internal_Agent* type. It relates to *this* agent.

The image below shows the Agents definition.

```
Agent: Internal_Agent | External_Agent ;

Internal_Agent: 'Internal_Agent'
            'Name' name=ID ','
            'That_Provide' provide=[Decrement_Event] ','
            'To_External_Participant' external_participant=[External_Agent] ','
            'By_Internal_Participant' internal_participant=[Internal_Agent] ';'

Internal_Agent: 'Internal_Agent'
            'Name' name=ID ','
            'That_Provide' provide=[Decrement_Event] ','
            'To_External_Participant' external_participant=[External_Agent] ','
            'By_Internal_Participant' internal_participant=[Internal_Agent] ';'
```

Figure 5.8: Agents

5.5.8 Contract - Production rule

The *Contract* action define several keywords and features:

- The rule starts to define two keywords *'Contract' and 'Type'* and a feature *name* which is assigned the value returned by rule *ID*. After each feature the non-terminal symbol *','* appear.
- The *'Internal_Party'* keyword and the feature *internal_party* are defined. It is a cross reference. This feature it's *Internal_Agent* type. It's the internal agent that participates in the *Contract*.
- The *'External_Party'* keyword and the feature *external_party* are defined. It is a cross reference. This feature it's *External_Agent* type. It's the external agent that participates in the *Contract*.
- The *'Clause_Increment'* keyword and the feature *increment_clause* are defined. It is also a cross reference. This feature it's *Increment_commitment* type. It represents a list of increment commitments indicated by the symbols "+=", the list can have zero or more elements indicated by the symbol "*". The elements of the list appear in parentheses and separated by white spaces.
- The *'Clause_Decrement'* keyword and the feature *decrement_clause* are defined. It is also a cross reference. This feature it's *Decrement_commitment* type. It represents a list of decrement commitments indicated by the symbols "+=", the list can have zero or more elements indicated by the symbol "*". The elements of the list appear in parentheses and separated by white spaces.

- The *'The_ Commitments'* keyword and the feature *commitment* are defined. It is also a cross reference. The feature it's *Commitment* type. It's a list of commitments that have been defined for this contract. Elements in the list are separated by commas.

The image below shows the Contract definition.

```
Contract: 'Contract'
        'Type' name=ID ','
        'Internal_Party' internal_party=[Internal_Agent] ','
        'External_Party' external_party=[External_Agent] ','
        'Clause_Increment' '('increment_clause+=[Increment_commitment]*')' ','
        'Clause_Decrement' '('decrement_clause+=[Decrement_commitment]*')' ','
        'The_Commitments' '('commitment+=[Commitment] (',' commitment+=[Commitment])*')' ';'
```

Figure 5.9: Contract

5.5.9 Commitment - Production rule

The *Commitment* action rule is defined as the choice ("|") between two types of actions (Increment_commitment and Decrement_commitment). The choice is modeled in the Ecore model by the relation *SuperType*.

5.5.10 Increment_commitment - Production rule

The *Increment_commitment* action define several keywords and features:

- The rule starts to define two keywords *'Increment_commitment' and 'Name'* and a feature *name* which is assigned the value returned by rule *ID*. After each feature the non-terminal symbol *','* appear.
- The *'Reciprocity_Relation'* keyword and the feature *reciprocity* are defined. It is a cross reference. This feature it's *Decrement_commitment* type. It represents the relation to a decrement commitment.
- The *'Reservation_For_Increment'* keyword and the feature *reservation_in* are defined. It is a cross reference. This feature it's *Resource* type. Represents the resource that will be incremented.

5.5.11 Decrement_commitment - Production rule

The *Decrement_commitment* action define several keywords and features:

- The rule starts to define two keywords *'Decrement_commitment' and 'Name'* and a feature *name* which is assigned the value returned by rule *ID*. After each feature the non-terminal symbol *','* appear.
- The *'Reciprocity_Relation'* keyword and the feature *reciprocity* are defined. It is a cross reference. This feature it's *Increment_commitment* type. It represents the relation to a increment commitment.
- The *'Reservation_For_Decrement'* keyword and the feature *reservation_out* are defined. It is a cross reference. This feature it's *Resource* type. Represents the resource that will be decremented.

```
Commitment: Increment_commitment | Decrement_commitment;

Increment_commitment: 'Increment_commitment'
                      'Name' name=ID ','
                      'Reciprocity_Relation' reciprocity=[Decrement_commitment] ','
                      'Reservation_For_Increment' reservation_in=[Resource] ';'

Decrement_commitment: 'Decrement_commitment'
                      'Name' name=ID ','
                      'Reciprocity_Relation' reciprocity=[Increment_commitment] ','
                      'Reservation_For_Decrement' reservation_out=[Resource] ';'
```

Figure 5.10: Commitments

5.5.12 Generate Language Artifacts

With the defined grammar it is necessary to create the various components of the language.It will generate a parser, serializer, and other infrastructure code. There are two ways to generate the necessary components:

- Right-click in the editor of file Reav3Dsl.xtext and select *Run As -> Generate Xtext Artifacts*.
- Locate the file GenerateReav3Dsl.mwe2 file next to the grammar file in the package explorer view. From its context menu, choose *Run As » MWE2 Workflow*.

The generated Ecore file, *Reav3Dsl.ecore* which has been generated from the grammar, can be found in the folder model/generated of project tmdei.rea.reav3. As written before, the ecore meta-model is the abstract syntax and the grammar is the concrete syntax. XText also produces a syntax graph from the menu [8] which allows to view the grammar with navigation support, double-click on an element in the graph diagram to jump to the corresponding part in the grammar syntax. The image below shows the partial grammar graph.

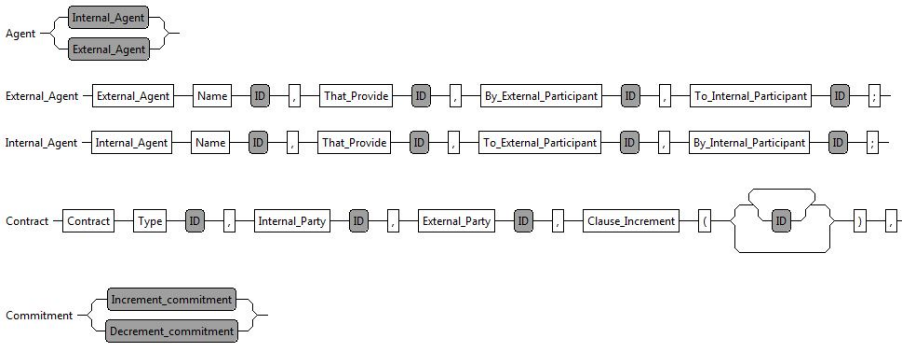


Figure 5.11: Partial Grammar Graph

It is also possible to create a class diagram from the *.ecore* file. From the context menu of the *Reav3Dsl.ecore* file, right-click and select *Initialize Ecore Diagram*.

[8] Window > Show View > Other > Xtext > Xtext Syntax Graph

5.6 DSL Usage

The DSL (tmdei.rea.reav3) will be extended by applying the DSL design pattern *Language Extension*. [9] This pattern is used to add new features to an existing language. The design of a DSL using this pattern involves the addition of new language elements to an existing base language. These elements can include new data types, language block interaction mechanisms, semantic elements, or syntactic sugar.5.6 In this case the DSL is extended with new elements: language interaction mechanisms and semantic elements. The process for the development of DSL has already been described. This section describes project necessary configuration and the new elements in *tmdei.rea.dsl4all*

5.6.1 Configuration

This section describes the settings required in the project *tmdei.rea.dsl4all* to extend the language *tmdei.rea.reav3*. The following steps are required:

- In the file *tmdei.rea.dsl4all\META-INF\MANIFEST.MF:* in the tab "Dependencies" under "Required Plug-ins" it is necessary to add two projects: tmdei.rea.reav3 and tmdei.rea.reav3.ui. This process has to be repeated for the files: *tmdei.rea.dsl4all.ui\META-INF MANIFEST.MF* and *tmdei.rea.dsl4all.ide\META-INF\MANIFEST.MF*.
- In the *Dsl4all.xtext* grammar file include the clause ("with") that declares this grammar is extended, like this:

  "grammar tmdei.rea.dsl4all.Dsl4all with tmdei.rea.reav3.Reav3Dsl"

  The image below shows the tab "Dependencies" and the project files added.

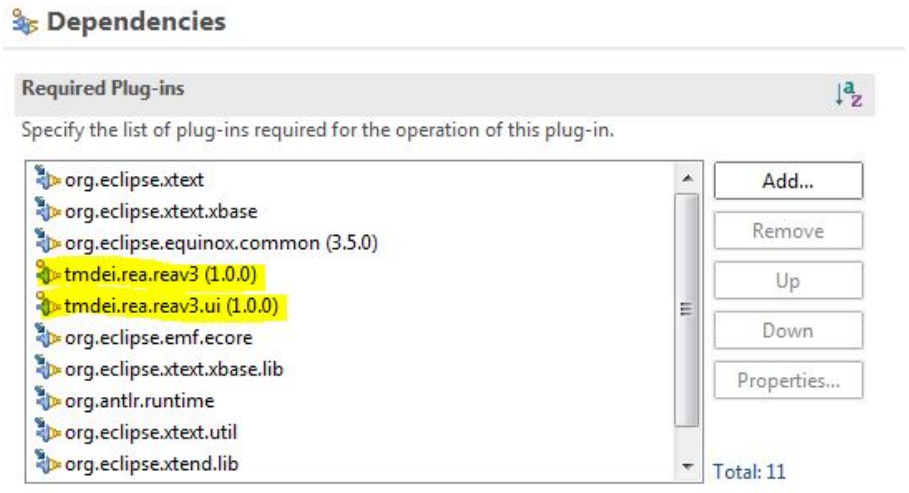


Figure 5.12: Tab "Dependencies" and the project files add

- In the file *tmdei.rea.dsl4all/src/tmdei/rea/dsl4all/GenerateDsl4all.mwe2*, block of code "language" it is necessary to place the file reference to *Reav3Dsl.genmodel*. The image below shows the required code.

[9] `https://www2.dmst.aueb.gr/dds/pubs/jrnl/2000-JSS-DSLPatterns/html/dslpat.html`

```
language = StandardLanguage {
    name = "tmdei.rea.dsl4all.Dsl4all"
    fileExtensions = "dsl4all"

    serializer = {
        generateStub = false
    }
    validator = {
        // composedCheck = "org.eclipse.xtext.validation.NamesAreUniqueValidator"
    }

    referencedResource=
        "platform:/resource/tmdei.rea.reav3/model/generated/Reav3Dsl.genmodel"
}
```

Figure 5.13: Referencing Resource - Reav3Dsl.genmodel

5.6.2 ReaModel - Production rule

The *ReaModel* action is the root node. It consists of a *Resources* list, an *Events* list, an *Agents* list and an optional feature *Contracts*.

- The feature *resources* represents a list of elements type *Resources* indicated by the symbols "+=".
- The feature *events* represents a list of elements type *Events*.
- The feature *agents* represents a list of elements type *Agents*.
- The feature *contracts* represents a list of elements type *Contracts*. This feature is optional indicated by the symbol "?".

5.6.3 Resources - Production rule

The *'Resources'* keyword and the feature *resource* are defined. The *resource* represents a list of elements type *Resource* indicated by the symbols "+=", the list can have zero or more elements indicated by the symbol "*". This block is placed between braces.

5.6.4 Events - Production rule

The *'Events'* keyword and the feature *event* are defined. The *event* represents a list of elements type *Event* indicated by the symbols "+=", the list can have zero or more elements indicated by the symbol "*". This block is placed between braces.

5.6.5 Agents - Production rule

The *'Agents'* keyword and the feature *agent* are defined. The *agent* represents a list of elements type *Agent* indicated by the symbols "+=", the list can have zero or more elements indicated by the symbol "*". This block is placed between braces.

5.6.6 Contracts - Production rule

The *'Contracts'* keyword and the features *contract* and *commitments* are defined. The *contract* list is followed by a list of *commitments* and this sequence may be repeated zero or more times, indicated by the symbol "*". This block is placed between braces.

5.6.7 Commitments - Production rule

The *'Contract_ Commitments'* keyword and the feature *commitment* are defined. The *commitment* represents a list of elements type *Commitment* indicated by the symbols "+=", The list must have at least one element indicated by the symbol "+". This block is placed between braces.

5.6.8 Instance Creation

From the meta-level workspace the language can be tested with a DSL instance creation. From the context menu, in the project *tmdei.rea.dsl4all*, right-click and select *Run As » Eclipse Application*. A new instance of Eclipse is executed with the necessary plug-ins for the new language. The image below shows a partial instance created in the language editor.

```
//Economic events (Exchanges) for the "Merchant of Venice"

Resources {Money, Ship, Employee Service, Silk}
Events { Increment Name Get Financing,
            Company_Take Get Financing,  //Increment event
            Company_Give Pay Financing, //Decrement event
            Duality_Relation Pay Financing, //Decrement event
            Company_Resource_Increment Money,
            Internally_Received_By Manager,  // Agent internal
            Event_Participants(Manager,Investor);
         Decrement Name Pay Financing,
            Company_Take Get_Financing,  //Increment event
            Company_Give Pay_Financing,  //Decrement even
            Duality_Relation Get_Financing,  //Increment event
            Company_Resource_Decrement Money,
            Externally_Received_By Investor, // Agent internal
            Event_Participants(Manager, Investor);
```

Figure 5.14: DSL Partial Instance

5.7 Model Transformation

This section describes the transformation of the model (generation of code) by creating a template, whose output is a Json file, according to the Numbersbelieve specifications for the OMNIA platform. As the image below shows, the transformation of the model is done at the meta level.

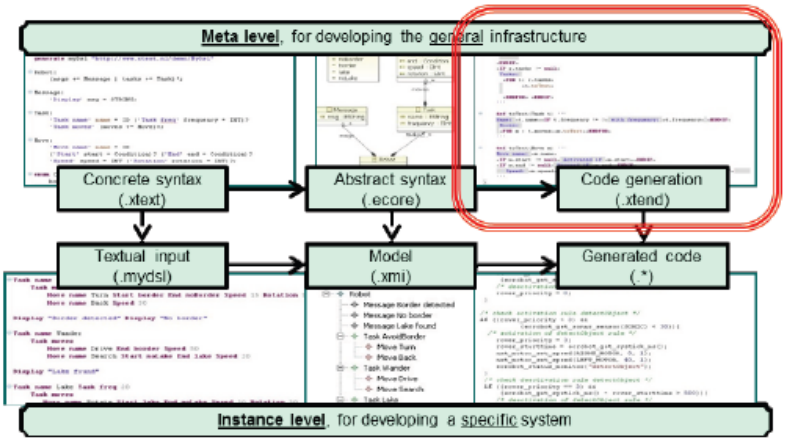


Figure 5.15: Model Transformation Level - source:5.4

5.7.1 Code Generation with XTend

Xtend is a programming language for the JVM, stays very close to Java and uses the exact same type system and it only depends a runtime library *Google Guava*. It also provides a series of *upgrades* relative to Java, some of this *upgrades* are: Type Inference, Extension Methods, Lambda Expressions and Template Expressions. Xtend has its roots in Xpand, a widely used code generation framework. [10]

5.7.2 File and Code Description

This class file *Dsl4allGenerator.xtend* [11] is used to generate code from the model in the standalone scenario and in the interactive Eclipse environment. The first task is to provide the model root element to which the transformation is applied. In this case this is done through the parameter "resource" invoking the "contents" function and assigning to the variable "reamodel", a Cast is made for list. Then you set the file, name, extension and location, to be generated. The image shows the complete code.

```
class Dsl4allGenerator extends AbstractGenerator {

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
    // Cast para uma Lista
    val reamodel=(resource.contents.head as ReaModel)

    fsa.generateFile(reamodel.class.simpleName+".xml", toXML(reamodel))
    fsa.generateFile(reamodel.class.simpleName+".json", toJson(reamodel))
```

Figure 5.16: Root element - "reamodel" and file setings

Once the root node and the file definitions have been defined, a series of *Template Expressions* are used to compose the generated code. *Template Expressions* are defined by the "def" statement followed by the function name and parameters if they exist. Enclosed by "', are defined expressions that are multi-line strings. These expressions can include others defined by *«expression»* used to create in-line code or call other functions. Xtend processes

[10] http://www.eclipse.org/Xtext/documentation
[11] tmdei.rea.dsl4all/src/tmdeirea/dsl4all/generator/Dsl4allGenerator.xtend

white spaces to generate properly formatted code. The image below shows the *Template Expression* that processes *Events* for Json.

```
//template expression para OMNIA events
def eventsOMNIAToJson(ReaModel rm)
'''
"events": [
«val evts=rm.events.head.event»
    «FOR evt : evts»
    «if(evt.name!=evts.last.name)'''«eventsOMNIAToJason(evt)»,'''
     else '''«eventsOMNIAToJason(evt)»'''»
    «ENDFOR»
],
'''

//template para OMNIA "events"
def eventsOMNIAToJason(Event evt)
'''
{
    "name": "«evt.name»",
    "attributes": [
        {
            "name": "_resource",
            "type": "«if(evt.eClass.name==="Increment_Event") evt.take.resource_increment.name else evt.give.resource_decrement.name»",
            «multyIsSystem»
        },
        {
            "name": "_provider" ,
            "type": "«if(evt.eClass.name==="Increment_Event") evt.give.receive.name else evt.take.receive.name»",
            «multyIsSystem»
        },
        {
            "name": "_receiver",
            "type": "«if(evt.eClass.name==="Increment_Event") evt.take.receive.name else evt.give.receive.name»",
            «multyIsSystem»
        }
    ]}
'''
```

Figure 5.17: Template Expression to processe Events to Json

5.7.3 Json file Creation

When a new instance is created, following the process in the section 5.6.8, and there is a transformation file, a new folder "src-gen" is created where the transformation files are placed. Whenever the instance is changed and saved the code generation is executed. The image below shows a partial result of the *Template Expression*, in section 5.7.2, in the Json file created.


```
"events": [
{
    "name": "Get_Financing",
    "attributes": [
        {
            "name": "_resource",
            "type": "Money",
            "multiplicity": {
                "lower": 1,
                "upper": 1
            },
            "isSystem": true,
            "aggregationKind": "shared"
        },
        {
            "name": "_provider" ,
            "type": "Investor",
            "multiplicity": {
                "lower": 1,
                "upper": 1
            },
            "isSystem": true,
            "aggregationKind": "shared"
        },
        {
            "name": "_receiver",
            "type": "Manager",
            "multiplicity": {
                "lower": 1,
                "upper": 1
            },
            "isSystem": true,
            "aggregationKind": "shared"
        }
    ]
```


Figure 5.18: Json - partial Events

5.8 DSL for OMNIA

After several tests using instances based on use cases provided by Numbersbelieve it was found that some of the features defined in the language contained *"duplicate"* data or data that could be inferred through another feature. This finding led the team to suggest that features with *"duplicate"* data or that it could be inferred from another feature be removed from the language. Two reasons were given: the former would be quicker to write data of an instance and the second because it would be easy to infer the data through model transformation. This has led to the development of a different DSL, the project *tmdei.rea.reav4*. This section describes changes to the initial language.

5.8.1 Increment_Event - Production rule

These are the concepts that are going to be removed:

- An Increment_Event is assigned to feature "take", the Increment_Event is the one being defined, for this reason the feature "take" is removed.
- An Decrement_Event is assigned to feature "give", Decrement_Event is the same being assigned to the feature "duality". The concept "duality" links two economic events together, and to remove this concept would be to lose that information, the decision is to remove the feature "give".
- An list of Agents is assigned to feature "participants". The list of Agents can be inferred through the relation "duality" and the concept "receive", for this reason the feature "participants" is removed.

5.8.2 Decrement_Event - Production rule

These are the concepts that are going to be removed:

- An Decrement_Event is assigned to feature "give", the Decrement_Event is the one being defined, for this reason the feature "give" is removed.
- An Increment_Event is assigned to feature "take", Increment_Event is the same being assigned to the feature "duality". The concept "duality" links two economic events together, and to remove this concept would be to lose that information, the decision is to remove the feature "take".
- An list of Agents is assigned to feature "participants". The list of Agents can be inferred through the relation "duality" and the concept "receive", for this reason the feature "participants" is removed.

5.8.3 Internal_Agent - Production rule

These is the concept that is going to be removed:

- An Internal_Agent is assigned to feature "internal_participant". The Internal_Agent is the one being defined, for this reason the feature "internal_participant" is removed.

5.8.4 External_Agent - Production rule

These is the concept that is going to be removed:

- An External_Agent is assigned to feature "external_participant". The External_Agent is the one being defined, for this reason the feature "external_participant" is removed.

5.8.5 Contract - Production rule

These is the concept that is going to be removed:

- To the feature "commitment" a list of Commitments is assigned, the list can be inferred through features "increment_clause" and "decrement_clause", for this reason the feature "commitment" is removed.

The image below shows the Increment_Event production rule of the original DSL and the modified DSL.

```
Increment_Event: 'Increment'
            'Name' name=ID ','
            'Company_Take' take=[Increment_Event] ','
            'Company_Give' give=[Decrement_Event] ','
            'Duality_Relation' duality=[Decrement_Event] ','
            'Company_Resource_Increment' resource_increment=[Resource] ','
            'Internally_Received_By' receive=[Internal_Agent] ','
            ('Fulfillment_in' '('fulfillment_in+=[Increment_commitment]*')'',')?
            'Event_Participants' '('participants+=[Agent] ',' participants+=[Agent] ')' ';'
```

Figure 5.19: Original Increment Event

```
Increment_Event: 'Increment'
             'Name' name=ID ','
             'Duality_Relation' duality=[Decrement_Event] ','
             'Company_Resource_Increment' resource_increment=[Resource] ','
             'Internally_Received_By' receive=[Internal_Agent] (',')?
             ('Fulfillment_in' '('fulfillment_in+=[Increment_commitment]*')')?';'
```

Figure 5.20: Modified Increment Event

5.9 Evaluation

The Evaluation is made according to the completion of the objectives, the functional and non-functional requirements. This evaluation in several aspects was done during the implementation phase. Together with the Numbersbelieve team, evaluation and quality characteristics were defined according to ISO/IEC 25010 [12], which defines several main characteristics. This characteristics were chosen to evaluate the artifacts: *Usability, Maintainability Satisfaction, Effectiveness, Efficiency, Portability, Functional Suitability*.

5.9.1 Functional Suitability

The objectives in terms of their conclusion: definition of the REA meta-model with the abstract syntax, the concrete syntax of the REA was defined through the language and

[12] https://www.iso.org/standard/35733.html

the automatic creation of the Json file for interaction with the OMNIA platform were all concluded.

5.9.2 Usability

ISO 9241-11 [13] defines Usability as *"The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*. In the context of REA DSL, the language is easy to learn, intuitive, effective and can be another tool that Numbersbelieve can offer its users to interact with the OMNIA platform .

- Effectiveness - in the context of OMNIA platform the effectiveness was total because the language and artifacts created (editor and automatic generation of Json) are suitable to solve the cases provided by Numbersbelieve for the OMNIA platform.
- Efficiency - through meetings and informal questionnaires it was found that the use of the language could be more efficient if some concepts, which were found to contain repeated data or that could be deduced from other present concepts, were withdrawn. Which led to the development of a new language with fewer concepts. The use of the new language proved to be more efficient in use but revealed to have less information available because now the missing information had to be deduced by the users and other stockholders.
- Satisfaction - there are two base languages that can be used and evaluated from the point of view of user satisfaction. The analysis of several test cases using the language revealed that the initial language satisfied the users (domain experts, stockholders) who needed the complete information to understand the processes and the new language was more efficient for the interaction with the OMNIA platform because it automatically created the Json file that was consumed by the platform API (no human read the data from the Json file). To establish a degree of satisfaction and efficiency with greater accuracy it is necessary to do a larger study and with more users and cases. Any of the languages completely complies with the functional and non-functional requirements, the plug-in created by XText is totally platform independent, allows syntax and semantic error detection, and context assistance with auto-complete.

5.9.3 Portability

The plug-in (language editor) created with XText is a completely independent application and can be used on any platform with the Java virtual machine installed.

5.9.4 Maintainability

The two base languages created are completely independent, any change in one language does not influence the other. The third language extends one of the two base languages and therefore changes in the extended language can interfere and may force changes.

[13] `https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en`

5.9.5 DSL Comparison

There is no formal or other evaluation with artifacts of the same nature (REA textual DSL) because it was not possible through literature analysis to find other artefact's.

Chapter 6

Conclusion

"Research has found that enterprise systems are consistent with REA Ontology. REA is actually an elementary set of concepts derived from basic definitions in accounting and economics" G. L. Geerts and McCarthy 2003, and this is the ISO standard [1] (ISO / IEC 15944-4: 2015) that incorporates the REA ontology for inter-organizational trading partners. Many efforts have been made to incorporate REA into information system development processes. At least two software applications in use today have used REA as a basis in the development process (`http://www.workday.com`) and (`http://www.reatechnology.com`).G. L. Geerts and McCarthy 2003

The development of the textual DSL helps in demonstrating that REA can be used to model the entire OMNIA platform. In a more general way the tool developed can help the promotion of REA in the modeling of information systems. It was verified that textual DSL can be used in a context of requirements analysis because both, business analysts and domain experts can use it. This can reduce the analysis time and increase the accuracy in the analysis of the processes.

Because REA concepts may not be fully understood by DSL users, the semantics used guide the description of processes. As concepts can only be related according to the relationships defined in the meta-model it can be assumed that users are not allowed to make mistakes. In this sense the learning is done quickly and concisely. Once the models are defined it is possible for software architects to transform models into other models and to develop architectures according to the models created. Since it is also possible to generate the necessary code for implementation, the idea of MDD is closer to be implemented.

Much effort was put into this document due to the amount of documents, technologies and software tools analyzed, the cost in terms of time was very high. However the result is innovative and adds a new tool that can be used by the community. All requirements were fully satisfied.

6.1 Limitations and Future Work

Although the developed DSL has been able to implement all the necessary requirements for the application of the OMNIA platform, it does not implement all REA concepts such as *Conversion, Policy, Type, Group, Schedule and others.*(Hruby 2013).

It was not possible to evaluate DSL semantics broadly, with a larger group and with domain experts from different areas of business

[1] `https://www.iso.org/obp/ui/#iso:std:iso-iec:15944:-4:ed-2:v1:en`

Also, a standalone application (DSL editor) was not created that could be sent to other teams and business environments for comments and evaluation.

From the limitations stated one could identify areas for future work:

- Create a standalone application with the textual DSL editor, to be easy to distribute to other teams and business environments.
- Implement more REA concepts.
- Evaluate semantics in different teams and business environments.

Bibliography

Al Jallad, Mohannad (2012). "Toward a REA based Domain Specific Visual Language". In: pp. 1–90. url: papers2://publication/uuid/49584C61-59F3-48A2-A590-F278DD8ABEA0.

Childs, Adam et al. (2004). "Cadena : An Integrated Development Environment for Analysis , Synthesis , and Verification of Component-Based Systems". In: Grant 11462, pp. 160–164. doi: 10.1007/978-3-540-24721-0{_}11. url: http://link.springer.com/10.1007/978-3-540-24721-0_11.

David, Julie Smith. "Three " Events " That Define an REA Methodology for Systems Analysis, Design, and Implementation Three " Events " That Define an REA Methodology for Systems Analysis, Design, and Implementation Three " Events " That Define an REA Methodology for Systems Analysis, Design, and Implementation INTRODUCTION". In: url: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.456.6800&rep=rep1&type=pdf.

EMFText (2012). "EMFText User guide". In:

Geerts, G. L. and W. E. McCarthy (2003). "An ontological analysis of the economc primitives of the extended-rea enterprise information architecture". In: *International Journal of Accounting Information Systems* 3.1, pp. 1–16. issn: 14670895. doi: 10.1016/S1467-0895(01)00021-5.

Geerts, Guido L and William E Mccarthy (2000). "The Ontological Foundation of REA Enterprise Information Systems". In: August, pp. 1–34.

Gruber, Thomas R (1993). *Substantial revision of paper presented at the International Workshop on Formal Ontology*. Tech. rep., pp. 907–928. url: http://tomgruber.org/writing/onto-design.pdf.

Hevner, Alan R et al. (2004). "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1, pp. 75–105. issn: 02767783. doi: 10.2307/25148625. url: https://pdfs.semanticscholar.org/fa72/91f2073cb6fdbdd7c2213bf6d776d0ab411c.pdf%20http://dblp.uni-trier.de/rec/bibtex/journals/misq/HevnerMPR04.

Hilwa, Al (2017). "The Rise of Low -Code Platforms and Modern Disruption". In:

Hruby, Pavel (2013). *Model-Driven Design Using Business Patternsnull*. Springer Berlin Heidelberg New York. isbn: 9783540301547. doi: 10.1108/k.2007.06736eae.002. url: http://dx.doi.org/10.1108/k.2007.06736eae.002.

Koen, Peter et al. (2001). "Providing Clarity and A Common Language to the "Fuzzy Front End"". In: *Research-Technology Management* 44.2, pp. 46–55. issn: 0895-6308. doi: 10.1080/08956308.2001.11671418. url: https://www.tandfonline.com/doi/full/10.1080/08956308.2001.11671418.

Merkle, Bernhard and Bernhard (2010). "Textual modeling tools". In: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion - SPLASH '10*. New York, New York, USA: ACM Press, p. 139. isbn: 9781450302401. doi: 10.1145/1869542.1869564. url: http://portal.acm.org/citation.cfm?doid=1869542.1869564.

O'Leary, Daniel E. (2010). "Enterprise ontologies: Review and an activity theory approach". In: *International Journal of Accounting Information Systems* 11.4, pp. 336–352. issn: 14670895. doi: 10.1016/j.accinf.2010.09.006. url: https://ac.els-cdn.com/S1467089510000722/1-s2.0-S1467089510000722-main.pdf?_tid=4d4c936a-1591-11e8-a55d-00000aab0f02&acdnat=1519057625_e5037168df61451c64e39e419ab78b8d.

OMG (2017). *Meta-Modeling and the MOF Meta-Modeling and the OMG Meta Object Facility (MOF)*. Tech. rep., pp. 1–7. url: http://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf.

Osterwalder, Alexander and Yves Pigneur (2011). *Business Model Generation - Inovação em Modelos de Negócios*. Vol. 3, p. 300. isbn: 9788576085508. doi: 65.012.2085b. url: http://brazil.enactusglobal.org/wp-content/uploads/sites/2/2017/01/Business-Model-Generation.pdf.

Pedro, Luis (2009). "EMFText How To : Developing an Organigram DSL using EMFText List of Figures". In:

Peffers, Ken et al. (2007). "A Design Science Research Methodology for Information Systems Research". In: *Journal of Management Information Systems* 24.3, pp. 45–77. issn: 0742-1222. doi: 10.2753/MIS0742-1222240302. url: http://www.tandfonline.com/doi/full/10.2753/MIS0742-1222240302.

Rymer, John R. (2017). *The Forrester Wave ™ : Low-Code Development Platforms For AD & D Pros , Q4 2017*. Tech. rep. url: https://reprints.forrester.com/#/assets/2/160/RES137262/reports.

Sonnenberg, Christian et al. (2011). "The REA-DSL: A Domain Specific Modeling Language for Business Models". In: pp. 252–266. doi: 10.1007/978-3-642-21640-4{_}20. url: http://link.springer.com/10.1007/978-3-642-21640-4_20.

Strembeck, Mark and Uwe Zdun (2009). "An approach for the systematic development of domain-specific". In: August, pp. 1253–1292. doi: 10.1002/spe.

Appendix A

DSL REA

The image in this appendix shows the grammar definition of the language referenced as "tmdei.rea.dsl4allv2.Dsl4allv2"

```
grammar tmdei.rea.dsl4allv2.Dsl4allv2 with tmdei.rea.reav4.Reav4Dsl

generate dsl4allv2 "http://www.rea.tmdei/dsl4allv2/Dsl4allv2"

// dsl4allv2  - Extension

ReaModel: resources+=Resources

             events+=Events

             agents+=Agents

             (contracts+=Contracts)?

;

Resources: 'Resources' '{' resource+=Resource ( ',' resource+=Resource)* '}'

;

Events: 'Events' '{' event+=Event  event+=Event* '}'

;

Agents: 'Agents' '{' agent+=Agent  agent+=Agent* '}'

;


Contracts:'Contracts' '{' contract+=Contract

    commitments+=Commitments  (contract+=Contract commitments+=Commitments)*


    '}'

;


Commitments: 'Contract_Commitments' '{' commitment+=Commitment+ '}'


;
```

Figure A.1: REA DSL - Grammar definition of "tmdei.rea.dsl4allv2.Dsl4allv2"

Appendix B

Intance_V1 - "Merchant of Venice"

The PDF in the appendix shows an instance of the "Merchant of Venice" scenario described in (David n.d.). The instance is based on the grammar "tmdei.rea.reav3.Reav3Dsl".

```
////Economic events (Exchanges) for the "Merchant of Venice"
Resources {Money, Ship, EmployeeServices, Silk}
Events { Increment Name Get_Financing,
            Company_Take Get_Financing,  //Increment event
            Company_Give Pay_Financing, //Decrement event
            Duality_Relation Pay_Financing, //Decrement event
            Company_Resource_Increment Money,
            Internally_Received_By Manager,  // Agent internal
            Event_Participants(Manager,Investor);
         Decrement Name Pay_Financing,
            Company_Take Get_Financing,  //Increment event
            Company_Give Pay_Financing,  //Decrement even
            Duality_Relation Get_Financing,  //Increment event
            Company_Resource_Decrement Money,
            Externally_Received_By Investor, // Agent internal
            Event_Participants(Manager, Investor);

          Increment Name Buy_ship,
            Company_Take Buy_ship,
            Company_Give Cash_Disbursement,
            Duality_Relation Cash_Disbursement,
            Company_Resource_Increment Ship,
            Internally_Received_By Manager,
            Event_Participants(Ship_Builder, Manager);
          Decrement Name Cash_Disbursement,
            Company_Take Buy_ship,
            Company_Give Cash_Disbursement,
            Duality_Relation Buy_ship,
            Company_Resource_Decrement Money,
            Externally_Received_By Ship_Builder,
            Event_Participants (Manager,Ship_Builder);

        Increment Name Employee_Service,
            Company_Take Employee_Service,
            Company_Give Pay_Salary,
            Duality_Relation Pay_Salary,
            Company_Resource_Increment EmployeeServices,
            Internally_Received_By Manager,
            Fulfillment_in (EmployeeService),
            Event_Participants (Manager, Employee);
        Decrement Name Pay_Salary,
            Company_Take Employee_Service,
            Company_Give Pay_Salary,
            Duality_Relation Employee_Service,
            Company_Resource_Decrement Money,
            Externally_Received_By Employee,
            Fulfillment_out (PaySalary),
            Event_Participants(Manager,Employee);

        Increment Name Purchase_Silk,
```

```
            Company_Take Purchase_Silk,
            Company_Give Pay_for_Silk,
            Duality_Relation Pay_for_Silk,
            Company_Resource_Increment Silk,
            Internally_Received_By Manager,
            Event_Participants(Manager, Silk_Seller);
        Decrement Name Pay_for_Silk,
            Company_Take Purchase_Silk,
            Company_Give Pay_for_Silk,
            Duality_Relation Purchase_Silk,
            Company_Resource_Decrement Money,
            Externally_Received_By Silk_Seller,
            Event_Participants(Manager,Silk_Seller);

        Increment Name Cash_For_Silk,
            Company_Take Cash_For_Silk,
            Company_Give Sell_Silk,
            Duality_Relation Sell_Silk,
            Company_Resource_Increment Money,
            Internally_Received_By Manager,
            Fulfillment_in (Cash_ForSilk),
            Event_Participants (Manager, Silk_Buyer);
        Decrement Name Sell_Silk,
            Company_Take Cash_For_Silk,
            Company_Give Sell_Silk,
            Duality_Relation Cash_For_Silk,
            Company_Resource_Decrement Silk,
            Externally_Received_By Silk_Buyer,
            Fulfillment_out (SellSilk),
            Event_Participants (Manager,Silk_Buyer);

        Decrement Name Penalty_Payment,
            Company_Take Cash_For_Silk,
            Company_Give Penalty_Payment,
            Duality_Relation  Cash_For_Silk,
            Company_Resource_Decrement Money,
            Externally_Received_By Silk_Buyer,
            Fulfillment_out (Penaltypay),
            Event_Participants ( Manager,Silk_Buyer);
        }


Agents { Internal_Agent Name Manager,
            That_Provide Pay_Financing,
            To_External_Participant Investor,
            By_Internal_Participant Manager;

        External_Agent Name Investor,
            That_Provide Get_Financing,
            By_External_Participant Investor,
```

```
            To_Internal_Participant Manager;

        External_Agent Name Ship_Builder,
            That_Provide Buy_ship,
            By_External_Participant Ship_Builder,
            To_Internal_Participant Manager;

        External_Agent Name Employee,
            That_Provide Employee_Service,
            By_External_Participant Employee,
            To_Internal_Participant Manager;

        External_Agent Name Silk_Seller,
            That_Provide Purchase_Silk,
            By_External_Participant Silk_Seller,
            To_Internal_Participant Manager;

        External_Agent Name Silk_Buyer,
            That_Provide Cash_For_Silk,
            By_External_Participant Silk_Buyer,
            To_Internal_Participant Manager;

}

// Business Events
Contracts {Contract Type Take_Order,
            Internal_Party Manager,
            External_Party Silk_Buyer,
            Clause_Increment (Cash_ForSilk),
            Clause_Decrement (SellSilk Penaltypay),
            The_Commitments(Cash_ForSilk,SellSilk,Penaltypay);

    Contract_Commitments { Increment_commitment Name Cash_ForSilk,
                                Reciprocity_Relation SellSilk,
                                Reservation_For_Increment Money;

                           Decrement_commitment Name SellSilk,
                                Reciprocity_Relation Cash_ForSilk,
                                Reservation_For_Decrement Silk;

                           Decrement_commitment Name Penaltypay,
                                Reciprocity_Relation Cash_ForSilk,
                                Reservation_For_Decrement Money;

    }
    Contract Type Employee_Salary,
        Internal_Party Manager,
        External_Party Employee,
        Clause_Increment (EmployeeService),
        Clause_Decrement (PaySalary),
```

```
        The_Commitments (EmployeeService, PaySalary);

    Contract_Commitments {Increment_commitment Name EmployeeService,
                          Reciprocity_Relation PaySalary,
                          Reservation_For_Increment EmployeeServices;

                         Decrement_commitment Name PaySalary,
                          Reciprocity_Relation EmployeeService,
                          Reservation_For_Decrement Money;
    }

}
```

Appendix C

Intance_V2 - "Merchant of Venice"

The PDF in the appendix shows an instance of the "Merchant of Venice" scenario described in (David n.d.) The instance is based on the grammar "tmdei.rea.reav4.Reav4Dsl"

```
////Economic events (Exchanges) for the "Merchant of Venice"
Resources {Money, Ship, EmployeeServices, Silk}
Events { Increment Name Get_Financing,
            Duality_Relation Pay_Financing, //Decrement event
            Company_Resource_Increment Money,
            Internally_Received_By Manager;  // Agent internal

          Decrement Name Pay_Financing,
            Duality_Relation Get_Financing,  //Increment event
            Company_Resource_Decrement Money,
            Externally_Received_By Investor; // Agent internal

          Increment Name Buy_ship,
            Duality_Relation Cash_Disbursement,
            Company_Resource_Increment Ship,
            Internally_Received_By Manager;

          Decrement Name Cash_Disbursement,
            Duality_Relation Buy_ship,
            Company_Resource_Decrement Money,
            Externally_Received_By Ship_Builder;

         Increment Name Employee_Service,
            Duality_Relation Pay_Salary,
            Company_Resource_Increment EmployeeServices,
            Internally_Received_By Manager,
            Fulfillment_in (EmployeeService);

         Decrement Name Pay_Salary,
            Duality_Relation Employee_Service,
            Company_Resource_Decrement Money,
            Externally_Received_By Employee,
            Fulfillment_out (PaySalary);

         Increment Name Purchase_Silk,
            Duality_Relation Pay_for_Silk,
            Company_Resource_Increment Silk,
            Internally_Received_By Manager;

         Decrement Name Pay_for_Silk,
            Duality_Relation Purchase_Silk,
            Company_Resource_Decrement Money,
            Externally_Received_By Silk_Seller;


         Increment Name Cash_For_Silk,
            Duality_Relation Sell_Silk,
            Company_Resource_Increment Money,
            Internally_Received_By Manager,
            Fulfillment_in (Cash_ForSilk);
```

```
        Decrement Name Sell_Silk,
            Duality_Relation Cash_For_Silk,
            Company_Resource_Decrement Silk,
            Externally_Received_By Silk_Buyer,
            Fulfillment_out (SellSilk);

        Decrement Name Penalty_Payment,
            Duality_Relation  Cash_For_Silk,
            Company_Resource_Decrement Money,
            Externally_Received_By Silk_Buyer,
            Fulfillment_out (Penaltypay);

        }


Agents { Internal_Agent Name Manager,
            That_Provide Pay_Financing,
            To_External_Participant Investor;


        External_Agent Name Investor,
            That_Provide Get_Financing,
            To_Internal_Participant Manager;

        External_Agent Name Ship_Builder,
            That_Provide Buy_ship,
            To_Internal_Participant Manager;

        External_Agent Name Employee,
            That_Provide Employee_Service,
            To_Internal_Participant Manager;

        External_Agent Name Silk_Seller,
            That_Provide Purchase_Silk,
            To_Internal_Participant Manager;

        External_Agent Name Silk_Buyer,
            That_Provide Cash_For_Silk,
            To_Internal_Participant Manager;

}

// Business Events
Contracts {Contract Type Take_Order,
            Internal_Party Manager,
            External_Party Silk_Buyer,
            Clause_Increment (Cash_ForSilk),
            Clause_Decrement (SellSilk Penaltypay);
```

```
    Contract_Commitments { Increment_commitment Name Cash_ForSilk,
                                Reciprocity_Relation SellSilk,
                                Reservation_For_Increment Money;

                           Decrement_commitment Name SellSilk,
                                Reciprocity_Relation Cash_ForSilk,
                                Reservation_For_Decrement Silk;

                           Decrement_commitment Name Penaltypay,
                                Reciprocity_Relation Cash_ForSilk,
                                Reservation_For_Decrement Money;

    }
    Contract Type Employee_Salary,
        Internal_Party Manager,
        External_Party Employee,
        Clause_Increment (EmployeeService),
        Clause_Decrement (PaySalary);

    Contract_Commitments {Increment_commitment Name EmployeeService,
                                Reciprocity_Relation PaySalary,
                                Reservation_For_Increment EmployeeServices;

                           Decrement_commitment Name PaySalary,
                                Reciprocity_Relation EmployeeService,
                                Reservation_For_Decrement Money;
    }

}
```