UNIVERSIDAD DE OVIEDO

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

"Extracción de un diagrama de clases UML a partir de requisitos en Esperanto utilizando Lenguaje de Dominio Específico y técnicas de Procesamiento del Lenguaje Natural"

DIRECTOR: Dr. Vicente García Díaz

AUTOR: Alberto Otero Márquez

Vº Bº del Director del Proyecto

Resumen

El análisis de requisitos es la etapa más importante de cualquier proceso de desarrollo software, ya que una recopilación incompleta de requisitos es la causa de que muchos proyectos de IT (Tecnología de la información) fracasen. Para mejorar el proceso de análisis de requisitos hemos desarrollado un sistema capaz de extraer un diagrama de clases UML (Lenguaje unificado de modelado) utilizando un documento de requisitos escrito en Esperanto. Hemos decidido emplear Esperanto,un lenguaje artificial que a efectos prácticas puede considerarse como lenguaje natural. A pesar de que la gramática del Esperanto es compleja y extensa, también es menos complicada, ya que no tiene excepciones. Debido a que su morfología es concisa y regular su procesamiento será más fácil. Distintas reglas heurísticas han sido empleadas para extraer los elementos del diagrama. Nuestro sistema ha generado con éxito un diagrama de clases con todas las clases relevantes relacionadas y muchas de las relaciones de asociación entre ellas, incluyendo relaciones de herencia y composición. Además, nuestro sistema es capaz de extraer relaciones de dependencia. Nuestro sistema ha sido validado utilizando dos casos de estudio, en el primero se ha comparado nuestro sistema con cinco diagramas obtenidos por expertos y con un diagrama generado por la herramienta RACE, y en el segundo, se ha comparado nuestro sistema con el diagrama creado por un experto con amplios conocimientos del dominio, demostrando que nuestro sistema es una verdadera alternativa.

Palabras clave: Técnicas de procesamiento del lenguaje; NLP, Ingeniería dirigida por Modelos; MDE, Lenguaje de Dominio Específico; DSL, Análisis de requisitos.

Abstract

Abstract Requirement analysis is the most important stage of any software development cycle, as incomplete requirement elicitation is often the reason many IT (Information technology) projects fail. In order to improve the requirement analysis process we have created a system that can extract a UML (Unified Modeling Language) class diagram using a requirements specification written in Esperanto. We opted to use Esperanto, an artificial language that may be considered as a natural language. Even though Esperanto's grammar is complex and extensive, it is also less difficult as it has no exceptions. Because its morphology is concise and highly regular it will facilitate its processing. Different heuristic rules have been used to extract the elements of the class diagram. Our system has successfully generated a class diagram with all the relevant classes implied and many of the association relationships between them, including also generalization relationships and composition relationships. In addition, dependency relationships have also been identified, and our system has been validated in two case studies, one by comparison with five experts' diagram and a diagram created by RACE, and the other by comparison with one expert's diagram with extensive domain knowledge showing that our system is a real alternative.

Keywords: Natural language processing; NLP, Model Driven Engineering; MDE, Domain Specific Language; DSL, Requirements processing

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my tutor at the University of Oviedo, Vicente Garcia, for the adecuate guidance and for being very patient and helpful.

Also, I would like to thank my friends for their participation in the testing, Javi, Pablo, Markitos and Diego.

Last but not least, thanks to my brother for his constant corrections and advice, and for always putting up with me.

1. Introduction

Natural language is the most frequently used language for expressing requirements, as it is common to customers, users and requirements engineers. Due to the ambiguity of natural language, which is liable for different interpretations influenced by geographical, psychological and sociological factors, a good requirement gathering process in natural language is usually difficult to perform Resnik et al. [1999]. To exemplarize the ambiguity consider the expression: "I saw a man on a hill with a telescope". Although it appears to be a simple statement, it may have many different meanings, like the following: (1) There is a man on a hill and I am watching him with my telescope. (2) There is a man on a hill, who I am seeing, and he has a telescope. (3) There is a man, he is on a hill that also has a telescope on it. (4) I am on a hill and I saw a man using a telescope. (5) There is a man on a hill, and I am sawing him with a telescope. Byrd [2009] An incomplete requirement elicitation is often the main reason that leads IT projects to failure. Projects with missing requirements will cause dissatisfaction to the customer, and incomplete products will require more time and resources to be finished than if originally planned Bergey et al. [1999]. A poor requirements analysis may produce the following effects Matsugu [2017]:

- Late product delivery: requirements are the draft that every member of a project works from. Poor requirements lead to poor designs and tests, and depending on when are these issues resolved, extra development and testing rework may need to be done. These issues, in addition with requirements revisions contribute to a late product delivery.

- Poor product quality: testing must be focused on the important parts of the project and must be as efficient as possible to perform the maximum amount of testing. The rework and the resulting wasted test runs reduce the overall amount and quality of testing that can be done. Depending on the quality of the requirements and wrong prioritization, the result can be poor product quality.

- Degraded design and documentation integrity: a series of changes take place as the flaws of dreadful requirements are discovered. This can degrade the design, lessen its integrity with each change. This affects not only the design itself, but it can spread to related assets like documentation.

- Invalid features delivered: for requirements flaws that are not uncovered or discovered become invalid features being delivered. These could be either features that are outside the scope of the project or incorrect features that will have to be handled by ad hoc solutions.
- Business impact: poor product quality, degraded design and documentation integrity and invalid features will have direct business impacts. The severity will vary depending on each given situation, but none of it will add value and they may be very serious. These elements impact customers, end users, business results, e.g. marketshare or profitability, and business aspects of aplication development and operation, including development costs, maintenance costs, administration costs, and it can ultimately lead to project failure or cancellation.

The importance of performing an exhaustive and complete requirement list on the early stages of a project is critical. Therefore, a support tool to automatize these tasks would be highly valuable and desirable. In recent years, Object-Oriented (O-O) Analysis and Design has become the mainstream trend for software development. A class is an abstraction of representative real-world objects. A class models the behavior of objects that are constrained by the same rules Starr and Stephen [2001] and share attributes and behaviors. Classes are arranged into a class diagram. A class diagram in the Unified Modeling Language (UML) represents the classes used and their relationships in the system. Those classes act as the vocabulary for O-O system, model simple collaborations and are the foundation for the logical database design Booch et al. [1999]. UML class diagrams are at the core of O-O analysis and identifying the classes that model the requirements seems crucial. In fact, it is considered by some authors as the most important skill in developing a O-O system Grady [1994]. Hence, the automated generation of class diagrams will save time and effort to a system analyst, specially for beginners. Nevertheless, the automation of class generation from a written text in natural language is extremely challenging due to the following reasons Richter [1999] Maciaszek [2001]:

- Natural languages are ambiguous and full of exceptions. Hence, rigurous and precise analysis proves quite difficult.
- Any given semantic can be represented in many different ways.
- Concepts that are not explictly expressed in the written source are often difficult to identify. It requires an expert domain knowledge to find the hidden classes.

Our approach in this paper tries to address the problem presented but it does not try to resolve it completely. We believe that by using Esperanto, an artificially made language with no grammatical exceptions, most part of the ambiguity would be suppresed. A Domain Specific Language (DSL) has been developed to model the most common behavior of the Esperanto language and act as a textual editor for the language. Natural Language Processing (NLP) techniques will be applied to the requirements document in Esperanto and different heuristic rules will be used to extract a UML Class diagram, which will contain the structure of a suggested system that satisfies the requirements. We have used NLP techniques and heuristic rules to address the following research questions: (1) How to identify concepts, (2) how to discern between candidate classes and attributes for each concept and (3) how to extract relationships between classes, such as generalization, dependency, associations and compositions.

2. Motivation

Several factors encouraged us to tackle a project based in Natural Language Processing (NLP) and Modeling:

- To improve the requirement analysis step, as it is the most important stage in any software development cycle. An incomplete or shallow elicitation requirements will lead to project failure, delays in the delivery date, increments in the project's budget or software that does not satisfy the expectations and needs of the customer. This has been an ongoing issue over the years, with not many research and solutions carried out to address it.

- General approach: Our project could be a potential help to any software project. Considering that every single project has some requirements to satisfy, our work could have a significant impact in the software development industry.

- There is an academic purpose besides helping in software developing. It can be a helping tool for new students to gain experience with requirements elicitation and UML class diagrams.

- The automatic generation of a class diagram containing the model that satisfies the requirements of a project could save both time and money. By making a complete and detailed requirement analysis we ensure that we have all the required information about: what we need to build, which type of users will use our product and other relevant details. Therefore, we could set up a good planning and optimize our resources. Furthermore, the final product should have a higher quality, as all the requirements should have been met, resulting a satisfied customer.

- The project presents many challenges and because the employed technologies have been barely or not seen at all during my Masters's studies, I believe it would complement well my education.

3. Contribution

This research intends to facilitate the whole software development process, specially in the requirement analysis stage. We intend to prove that a Model Driven Approach (MDA) can be applied in conjunction with NLP techniques to address this issue. We have provided the following advantages:

- Domain Specific Language (DSL) that contains the behavior of the Esperanto language, which will be the language used for the user to define the requirements that our system will process to generate a UML class diagram. This was the selected approach as it is the most beneficial both for the users, providing them with a textual editor, detailed feedback through syntax highlighting and an outline view, and for the system due to an automatically mapping process from the words that compose the requirements document provided by the users to Objects processed by our system.

- Ease of use for the users: As a language is possibly the most expressive communication tool that we have as a species, considering that the users simply have to write using a language (Esperanto) the different requirements for their project, we believe it will simplify the process.

- Transparency for the users: our system will automatically generate a UML class diagram as soon as the user saves the requirements using the provided editor. The users will not become aware of the underlying technology that processes and generates said diagrams.

- Automatic generation of a UML class diagram by processing a requirements document written using an (artificial) language (Esperanto). The diagram generated will be composed by the different classes, attributes and relationships (association, composition, generalization and dependency relationships) between the classes that represent a possible solution to the given problem.

4. Objectives

The main objectives of this project are:

- Develop an interdisciplinary project combining different fields such as Model Driven Architecture (MDA), Natural Language Processing (NLP), Modeling and Unified Modeling Language (UML).
- Model the most common type of grammatical structures and vocabulary in Esperanto by developing a Domain Specific Language (DSL).
- Apply different NLP techniques and heuristic rules to extract the elements of a UML class diagram from a written requirements document.
- Generate a valid UML class diagram by processing a requirements document.
- Provide the users with a customized text editor that would be used to enter the requirements of their project.
- Develop a system easily extendable and maintainable by using MDE.

5. State of the Art

In this section we have gathered the most recent advances in three different areas. First, we will discuss different approaches employed in natural language processing (NLP). Then, we will present the possible advantages of using an intermediate language in place of widely extended languages, such as English. Finally, we will explore the existing technologies employed when dealing with Model Driven Architecture (MDA).

5.1 Natural Language Processing to extract some Unified Modeling Language (UML) diagram

We have identified two different type of solutions when using natural language processing to generate Unified Modeling Language (UML) diagrams (use case diagram, class diagram and so on) : (1) those which employ heuristic rules or guidelines to perform NLP, and (2) those with machine learning algorithms or deep learning with NLP.

Next, there is a list of the different approaches and achievements made by different authors using heuristic rules and guidelines:

- Ibrahim and Ahmad [2010] proposed the Requirements Analysis and Class diagram Extraction (RACE) method, which combines the use of NLP techniques and domain ontologies to extract a class diagram from a given set of informal requirements. The components used in this system are: (1) OpenNLP as a parser, (2) a Stemming Algorithm; responsible of abbreviating words by removing affixes and suffixes, (3) WordNet; in charge of providing semantic validation during the process, (4) domain specific Ontology Library; to enhance the concept's identification process and (5) Class Extraction Engine, which uses heuristic rules to extract the class diagram. The results obtained in their case study are promising. The system was able to find concepts based on nouns, verbs and noun phrases as well as identifying four different types of relationships: Generalization, Association, Composition, and Aggregation. Nevertheless, it failed to identify one to one, one to many, and many to many relationships.
- Deeptimahanti and Babar [2009] developed UML Model Generator from Analysis of Requirements (UMGAR), a tool capable of generating Use-case diagrams, analysis and design class model and collaboration

diagrams by processing requirements expressed in natural language, doing it in an automatized way. The key components of this system are: (1) Stanford Parser; used to extract information from each requirement, avoiding the use of different single components for tasks such as stemming and tagging tools, (2) WordNet; which provides morphological analysis and plural to singular conversion and (3) JavaRAP; which provides pronoun's treatment. Ultimately, UMGAR is a domain independent tool capable of generating different UML models from requirements expressed in natural language, offering also XML support for the visualization of the diagrams created.

- Letsholo et al. [2013] designed Textual Requirements into Analysis Models (TRAM), a tool able to assist in the automatic creation of analysis models from requirements expressed in natural languages. It offers three advantages over similar existing tools, (1) it closes the gap between unstructured natural language requirements and its analysis models by providing a series of conceptual patterns, named Semantic Object Models (SOMs), (2) it offers requirements traceability to assist analysts and developers to locate errors and (3) it validates and improves the quality of the model generated by the domain expert or software analyst by using a question and answer method.

- Zhou and Zhou [2004] proposed to use NLP techniques to handle the written requirements and domain knowledge through ontologies to improve the performance of class identification. Their designed system used two inputs: (1) text functional specifications and (2) structured domain ontologies. Its output is a class diagram, which includes attributes of each class and inter-class relationships. They were able to identify the following relationships; generalization, aggregation and association. Furthermore, they classified the association relationships into one-to-one, one-to-many and many-to-many. The core components of the system are: (1) Transformation Tagger for part-of-speech tagging, (2) Link Grammar as the sentence parser and (3) WordNet for semantic validation. They outlined a possible improvement in the diagrams generated by employing ontologies.

- Song et al. [2004] proposed a Taxonomic Class Modeling (TCM) methodology. It was able to apply noun analysis, class categorization, sentence structure rules, checklists and heuristic rules for modeling. In their approach, they were able to identify: (1) classes from concepts stated as noun phrases in the requirements statement, (2) classes stated as a verb phrase in the requirements statement and (3) hidden classes that were not explicitly stated in the requirements statement by applying domain knowledge to the class categories. They also employed WordNet as many of the works previously presented. They found that the TCM methodology is effective to identify domain classes for object-oriented applications in different domain fields. They state that their solution is practical as it can be easily and effectively applied to different project domains.

We have also analyzed relevant works made using machine or deep learning in conjunction with NLP:

- Collobert et al. [2011] developed a unified neural network architecture and learning algorithm to be applied to NLP tasks, such as part-of-speech tagging, chunking, entity recognition and semantic role labeling. They used the entire English Wikipedia, containing about 631 million words, to train their algorithm. They developed SENNA (Semantic/syntactic Extraction using a Neural Network Architecture), a standalone version of their architecture written in C. Based on their benchmarks against state of the art systems, they managed to build a tagging system with a good performance and minimum computational requirements.

- Morwal and Chopra [2013] developed Name Entity Recognition using Hidden Markov Model (NERHMM), which is able to perform name entity recognition in natural languages. This system takes pre-labeled data as input and generates a Start Probability, a Transition Probability and an Emission Probability. They also implemented the Viterbi algorithm that returns the optimal state sequence. Their results show that with a lot of training they can obtain high accuracy, up to more than 90% accuracy .

- dos Santos and Zadrozny [2014] proposed a Deep Neural Network (DNN) architecture to join word-level and character-level representations to perform part-of-speech (POS) tagging. They developed two POS taggers, one for English and one for Portuguese, with accuracies of 97.32 % and 97.47% respectively.

- Gillick [2009] proved the feasibility of using support vector machines for Sentence Boundary Detection (SBD). They obtained inspiring results in more than 26000 examples in English, with a 0.24% error rate.

Despite this low error rate, their work is not complete, as there are remaining cases that require further work, like abbreviations.

5.2 Why Esperanto?

Originally, English was considered for our project, but it implied the use of multiple systems to perform parsing, stemming, semantic validation and more,which are not available for Esperanto. By using Esperanto external components are not required, as the parsing and other related tasks will be performed by ourselves.

Besides, there are a couple more reasons that make Esperanto the most preferable choice as the language to work in this project:

1. To apply traditional parsing a complete formal grammar of the language is required, being context-free and unambiguous. Natural languages, like English or Spanish, do not meet those requirements. These languages are complex and full of exceptions. Esperanto can be considered as a natural language for our purposes, as it includes all the common word types and grammatical features. Even though Esperanto's grammar is complex and extensive, it is also less complicated due to a complete lack of exceptions. Its morphology is concise and highly regular, which will facilitate its parsing Aasgaard [2006].

2. The basics of the Esperanto grammar are covered by 16 rules, which simplify and speed up the learning and processing of the language Forster [1982].

There are also some disadvantages by using Esperanto, namely:

1. Esperanto is not widely extended, being spoken or understood by a few millions of people.

2. Only users with knowledge of Esperanto would be able to use our system.

In addition, some parsers have been already developed for Esperanto. They take advantage of the highly regular morphology of Esperanto, its design to avoid ambiguity and the total lack of exceptions. These parsers are EspGam Bick [2007] and EOParser GermaneSoftware [2006].

EspGam is a Constraint Grammar parser for Esperanto. It is a rule based system that applies contextual rules to tackle morphological disambiguation and perform syntactic analysis. The system contains a lexicon of

28.000 lexemes built from data of a bilingual Esperanto-Danish dictionary and a Danish-Esperanto machine translation system. The disambiguation and syntactic rules are formulated by removing, selecting or mapping category tags, based on sentence-wide context conditions. They measured the performance of their parser against a hand-annotated standard corpus of news text produced in Esperanto. The parser accuracy rates are of 99.5% for POS and 92,1% for syntactic dependency.

EOParser is a morphology parser written in Ruby for Esperanto. EOParser offers a text-based User Interface for querying, and it can also be used as a library. The parser accepts Esperanto sentences and returns the translation as a string based output and the morphological properties as long as it understands the semantic meaning of the query.

5.3 Putting it all together: MDE, MDA and DSL

5.3.1 Model Driven Engineering

Model Driven Engineering (MDE) is a bright approach to address platform complexity and to provide the means to express domain concepts adequately by the developers. MDE technologies aim to merge domain-specific modeling languages with transformation engines and generators. The domain-specific modeling languages let developers define the relationships among concepts in a domain, defining the semantics and constraints associated with these domain concepts in a proper way. Then, the transformation engines and generators would synthesize different artifacts, such as source code, deployment descriptors or alternate model representations, which will ensure consistency between application implementation and analysis information related with functionality and requirements.

In addition, the learning curve is flattened due to the presence of graphic elements that relate to familiar domains, helping system engineers and software architects to ensure their software meet the user needs. Furthermore, MDE tools enforce domain-specific constraints and perform model checking that can detect and prevent many errors in the early life cycle of the software Schmidt [2006].

MDE is supposed to increase productivity by maximizing compatibility between systems (by reusing standardized models), simplifying the design process (models with recurring design patterns in the domain) and improving communication between teams working on the system (by terminology standardization and best practices used in the application domain).

Two of the most relevant MDE initiatives are:

- Model Driven Architecture (MDA): provides a set of guidelines for model-driven development using Object Management Group (OMG) technologies.

- Eclipse Modeling Framework (EMF): supports the key MDA concept of using models as the input for development and integration tools.

5.3.2 Model Driven Architecture

Model Driven Architecture (MDA) is a software design approach based on the Unified Modeling Language (UML) and other standards for storing, visualizing and exchanging software designs and models. MDA encourages the creation of machine-readable, highly abstract models developed independently of the implementation technology and stored in standardized repositories, where they can be accessed repeatedly and automatically transformed into schemes, code skeletons, test cases and deployment scripts for different platforms Kleppe et al. [2003].

MDA has three main objectives: (1) portability (2) interoperability and (3) reusability. MDA promises to overcome the challenges in highly interconnected and changing systems, regarding business rules as well as technology, offering a framework that provides the following:

- Portability: increasing the reutilization of existing applications and reducing the cost and complexity of the development and maintenance of applications.

- Platform interoperability: using strict methods to ensure that the standards based in different technology implementations all have the same business rules.

- Platform independence: reducing the time, cost and complexity associated with applications deployed using different technologies.

- Productivity: allowing developers, designers and system administrators to use languages and concepts that are familiar to them, facilitating the communication and integration between work teams.

MDA specifies three basic models that compose a system, where each of them can be considered as a level of abstraction in which different models can be built.

These models are Miller et al. [2003]:

1. A Computation Independent Model (CIM) is a view of the system from the computation independent viewpoint. It does not contain details of the structure of the systems. Sometimes called a domain model, it uses a vocabulary familiar to the practitioners of the specific domain used to its specification. The main user of the CIM is the domain expert or business consultant. It specifies the functionality of a system without considering construction details. It plays an important role in closing the gap between domain experts and experts of the design and construction of the system.

2. A Platform Independent Model (PIM) is a view of the system from the platform independent viewpoint. A PIM presents an specified degree of platform independence in order to be suitable for different platforms of similar type. It describes the construction of a system from an ontological level, specifying the system construction without implementation details. A ontological model Dietz and Hoogervorst [2007] should have the following properties: (1) composition: a set of elements of some category, (2) an environment: a set of elements of the same category, (3) production: the elements in the composition produce products or services that are delivered to the elements in the environment and (4) a structure: a set of interactions between the elements in the composition, those in the structure and those in the environment.

3. A Platform Specific Model (PSM) is a view of the system from the platform specific viewpoint. A PSM combines the specifications defined in the PIM with the details that explain how that system uses a particular type of platform. A PSM is a more detailed type of PIM, where platform specific elements are included. A target platform is necessary for the definition of a PSM.

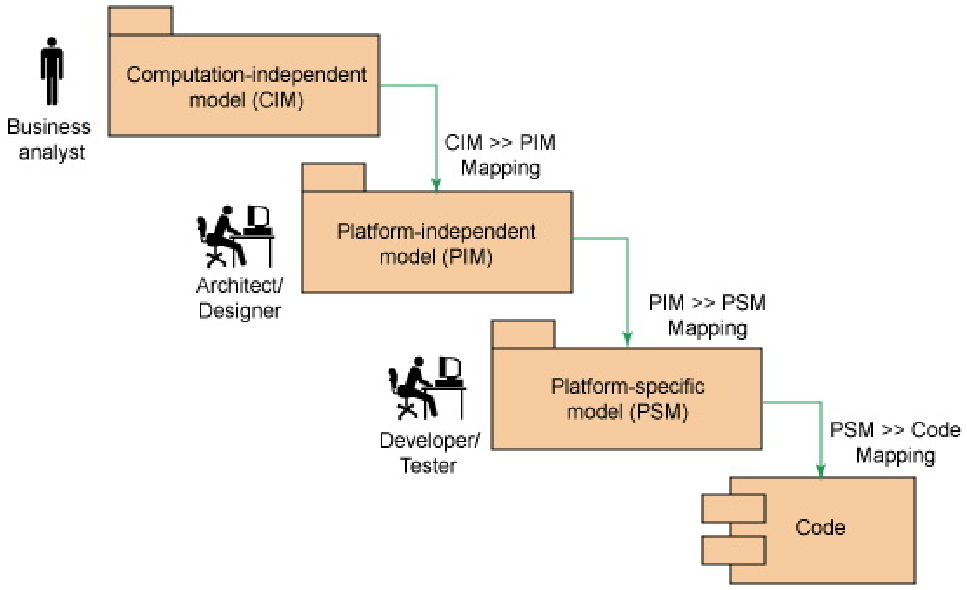


Figure 5.1: Model Driven Development. MaheshH.Dodani [2006]

In figure 5.1 we can observe how the CIM does not include any system details as it is defined by business requirements. Design experts connect new technologies with artifacts meeting the domain requirements, which yields a PIM. Once this PIM is completed, using a transformation process based on the target implementation platform the PSM is generated.

5.3.3 Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) is a framework and code generation tool for building Java applications based on simple model definitions. EMF unifies three technologies: Java, XML and UML. EMF offers a runtime framework that allows modeled data to be easily validated, persisted and edited in a User Interface (UI). Metadata is useful for enabling generic processing of any data using a uniform and reflective API. One of the most relevant aspects of EMF is its interoperability with other EMF-based tools and applications Steinberg et al. [2008].

The model used to represent models in EMF is Ecore, which is a simplified subset of UML. The basic elements of Ecore are:

1. Eclass: used to represent a modeled class. It has a name and can contain attributes and references.

2. EAttribute: employed to represent a modeled attribute. It has a name and a type.

3. EReference: represents one end of an association between classes. It has a name, a reference type, which is another class, and a boolean that indicates whether it represents containment or not .

4. EDataType: indicates the type of an attribute. It can be a primitive type, such as an int or float, or an object type like java.util.Date.

EMF helps closing the gap between modelers and Java programmers. Models can be defined using UML by an XML Schema or by using annotations on Java interfaces. Given a model, EMF allows the user to generate any of the other artifacts and the implementation classes that represent the model.

5.3.4 Domain Specific Languages

A Domain Specific Language (DSL) is a programming language or executable specification language that offers expressive power usually restricted to a particular problem domain through appropiate annotations and abstractions Van Deursen et al. [2000]. Some relevant and widely known DSL languages are SQL, BNF and HTML.

Adopting a DSL approach to software engineering involves risks and opportunities, some of the later being:

- Problems can be expressed with a precise language and a level of abstraction such as the problem domain. Then, domain experts understand, validate, modify or develop DSL programs.

- DSL programs are concise, largely self-documented and can be reused for different purposes.

- DSLs enhance productivity, reliability, maintainability and portability.

- DSLs express domain knowledge, and thus enable the conservation and reuse of this knowledge.

- DSLs allow validation and optimization at the domain level.

The disadvantages of the use of a DSL are:

- The costs of designing, implementing and maintaining a DSL as well as the costs of educating DSL users.

- The limited availability of DSLs, the difficulty of finding the proper scope for a DSL and balance between domain-specificity and general-purpose programming language constructs.
- The potential loss of efficiency with respect to hand-coded software.

There are different kinds of DSLs syntactically, the most used being either **graphical**, such as Sirius Foundation [2007a] or **textual** lik Xtext Foundation [2006].

5.3.4.1 Sirius

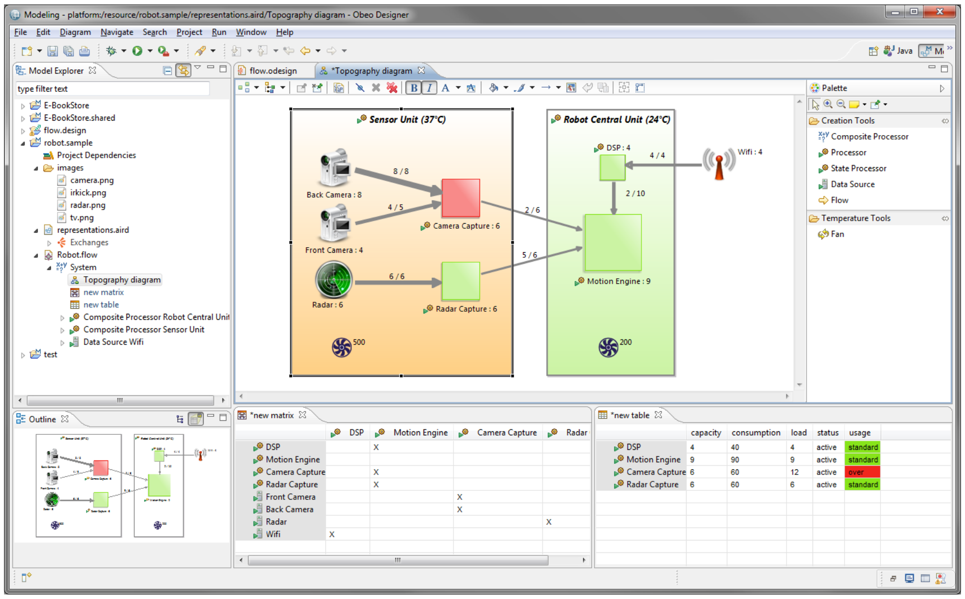
Sirius is a framework built on top of a Graphical Modeling Framework(GMF) that allows to create, visualize and edit models using interactive editors.

Sirius provides different viewpoints to perform analysis and look at the different roles on the same domain model. Sirius provides a graphical modeler to create a Domain Specific Model (DSM), which defines concepts and their relationships within the model. It also allows for an easy transformation of a DSM to a specific representation of the model, offering different representations such as diagrams, tables, cross-tables or trees. The user can create, modify and validate his designs through these representations Viyović et al. [2014].

Sirius' strengths are:

- Adaptability to any EMF compatible DSM.
- Strong separation between semantic and representation models.
- Support for different representations of a domain model.
- Based on an open and widely used industry standard, EMF.
- Easy to use and rapid development.
- High level of extensibility.

In figure 5.2 we can observe an implementation using Sirius's graphical editor of a system employing cameras, robots and different sensors.



*new matrix

	DSP	Motion Engine	Camera Capture	Radar
DSP		X		
Motion Engine				
Camera Capture		X		
Radar Capture		X		
Front Camera			X	
Back Camera			X	
Radar				X
Wifi	X			

*new table

	capacity	consumption	load	status	usage
DSP	4	40	4	active	standard
Motion Engine	9	90	9	active	standard
Camera Capture	6	60	12	active	over
Radar Capture	6	60	6	active	standard

Figure 5.2: Sirius Modeling Perspective Foundation [2007a]

5.3.4.2 Xtext

Xtext is a framework designed to quickly develop tools for textual languages. These languages range from small Domain Specific Languages (DSL) to General Purpose Languages (GPL), covering textual configuration files and requirement documents written in natural languages. The benefits of choosing the appropriate tool are an improvement in readability, writability and understandability of documents written in those languages.

Xtext first defines a grammar, then generates a parser, serializer and a smart editor for that language. Even all the generated artifacts can be customized through dependency injection. Xtext provides an easy-to-use API for tasks that need customization, such as validation or linking/scoping. Xtext relies in the Eclipse Modeling Framework (EMF), with the Abstract Syntax Tree (AST) generated by Xtext's parser being an Ecore model (refer to section 5.3.3). The correspondent Ecore model can be either derived from the grammar or specified explicitly. In consequence, Xtext allows an easy integration with Eclipse Modeling environment tools, such as Model-to-Model or Model-to-Text transformation languages Eysholdt and Behrens [2010].

In the Eclipse environment, the Graphical Modeling Framework (GMF) is a tool that allows developers to easily define graphical editors for EMF-based models. Graphical editors are usually not powerful enough, as many problems are better described through textual specifications.

We will now we present different applications and solutions that have been accomplished thourgh Xtext:

- Behrens [2010] proposed an implementation of a model-based solution using a DSL to describe the structure and behavior of mobile applications. They provided the tools to support static analysis, code navigation, as well as compiler and simulator integration of the iPhone development platform. They built a textual DSL in conjunction with a complementary code generator to provide inexperienced developers in Objective-C and the underlying API provided by the iPhone SDK (iOS SDK) with tools to support the developers' learning process.
  They chose Xtext to develop the DSL and to implement various validation rules. They benefited from Xtext's productivity features, such as content assist, code templates and version control integration. In top of that, they contributed with the following additional features: (1) symbolic integration: providing content assist, error markers for unknown files and quick fixes in case of typing errors for external resources, (2) incremental build: through a background service that regenerates artifacts as needed, so it can transparently trigger other compilers and builders after code's execution and (3) invocation of the iPhone simulator: the launch of the iPhone simulator from the DSL editor is similar to launching a Java process from a Java file. The code generator was developed with Xpand. The Eclipse platform was used to implement incremental code generation, compilation of the generated Objective-C code and invocation of the iPhone simulator provided by the Apple SDK without the need to manage external tools manually. The developers were able to implement additional views and test them inside the Eclipse based development environment. The full development cycle of creating, testing and changing applications is covered with this tool chain.

- Heitkötter et al. [2013] proposed md^2; an approach for model driven cross-platform development of apps. Following this approach, developers define the behavior of an app in a high-level (domain-specific) language designed for describing business apps in a short and brief way. From the model, native apps for Android and iOS are automatically generated and are able to access the device's hardware and provide a native look and feel. Xtext was used to construct the abstract syntax and the textual notation of the language. The editor provided by Xtext offers desirable features, such as syntax highlighting, content assistance and validation, allowing app developers to quickly specify the model of their app and help them during modeling. The code generation is automatically run once the developer saves his model. The model is preprocessed to simplify subsequent generation, then the Android and iOS code generator generates the apps for each platform translating the model into source code. Java code is generated for Android, as well as Objective-C code for iOS. The generator also creates XML files to include a graphical user interface (GUI) in Android and to specify the data model in iOS. Furthermore, project files and settings are generated for Eclipse (Android) and Xcode (iOS). md^2 proved to be a suitable model-driven cross-platform app development tool for typical business applications for mobile devices.

- Conejero et al. [2015] proposed a Model-Driven approach to develop video-surveillance applications. They proposed this solution to deliver very reusable and highly configurable systems. They developed VideoDSL; a DSL to define video-surveillance applications in a high level of abstraction. These applications can be defined using a graphical editor or a textual editor. The models defined using their DSL automatically create the configuration of the applications. Their approach provides different versions of the system and a standalone, web and mobile applications. Xtext is used to define the DSL in a textual way and perform different validations applied to the grammar defined in Xtext to ensure that certain restrictions are enforced. They also used Xpand to generate SQL code that stored the needed information in their database. All these processes occur in model to code transformations.

- Funk and Rauterberg [2012] developed PULP, a DSL that provides abstraction from asynchronous JavaScript, state machines and access to cross-platform media playback, which is generated in a final model-to-text transformation. In addition, the DSL allows tying content such as images and media files together by modeling the dynamic behavior, movements and control flow. They used Xtext to provide a textual editor with syntax highlighting, auto completion and validation. The code generation is performed using Xtend, considered to be the successor of Xpand. The PULP generator for HTML5 runs automatically when a file is saved in the editor, creating a hierarchy of files including the images and media related. Lastly, JavaScript code is generated according to the given interaction capabilities and defined behavior of objects. PULP proved to be suitable for rapid prototyping of web-based games.

6. Domain Specific Language

We developed a Domain Specific Language (DSL) to implement the most common structures of the language Esperanto. We decided to use Xtext Foundation [2006] as it is one of the most common tool employed to create textual DSLs. That way we could benefit from its features, such as the generation of a parser, serializer and a smart editor. The DSL that we developed is actually a textual editor for the Esperanto language, to be more precise, to a big subset of the Esperanto language. Another reason to use Xtext is the fact that Xtext uses and integrates with the Eclipse Modeling Framework Steinberg et al. [2008] (see section 5.3.3) , a framework and code generation tool for building Java applications based on model definitions. We provided further customization for the smart editor, so the user would feel more comfortable when typing his requirements. In that way, we have unified three technologies, Java, XML and UML.

Model Driven Engineering and Domain Specific Languages are complementary and necessary for model-driven approaches. MDE is used to define the type of models we need to describe a software application. DSLs are used to define languages for expressing the models. MDE allows to define in a more precise way the modeling dimensions. This is relevant because one of the biggest drawbacks of DSL design is that the scope grows out of proportion, due to changing requirements, and the result is that the DSL becomes a GPL. den Haan [2009].

In our project, by using MDE and a DSL, changing the underlying technologies that the system depends on is a more much easier process than for the majority of traditional systems. We believe this to be a huge advantage of our system.

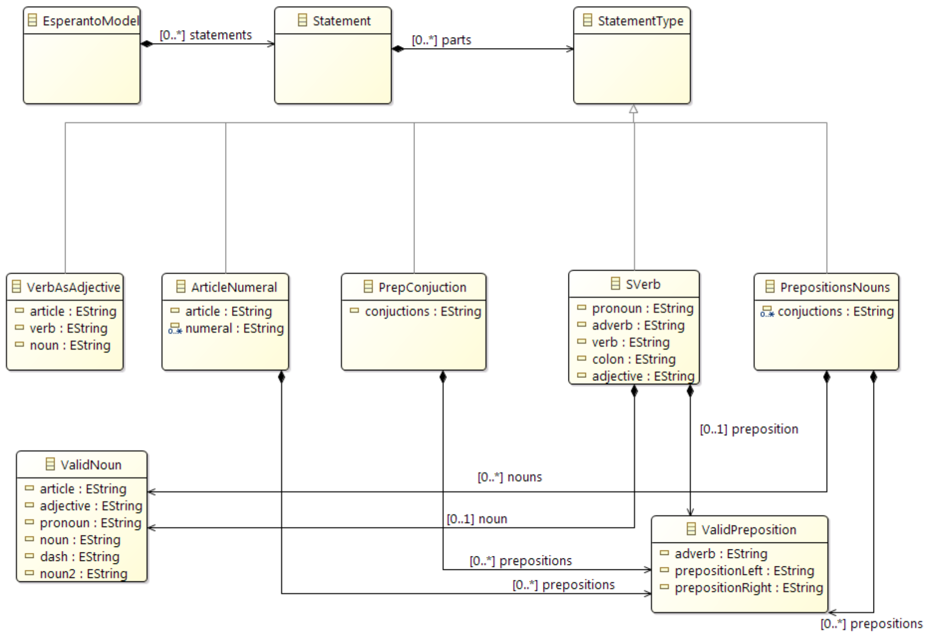
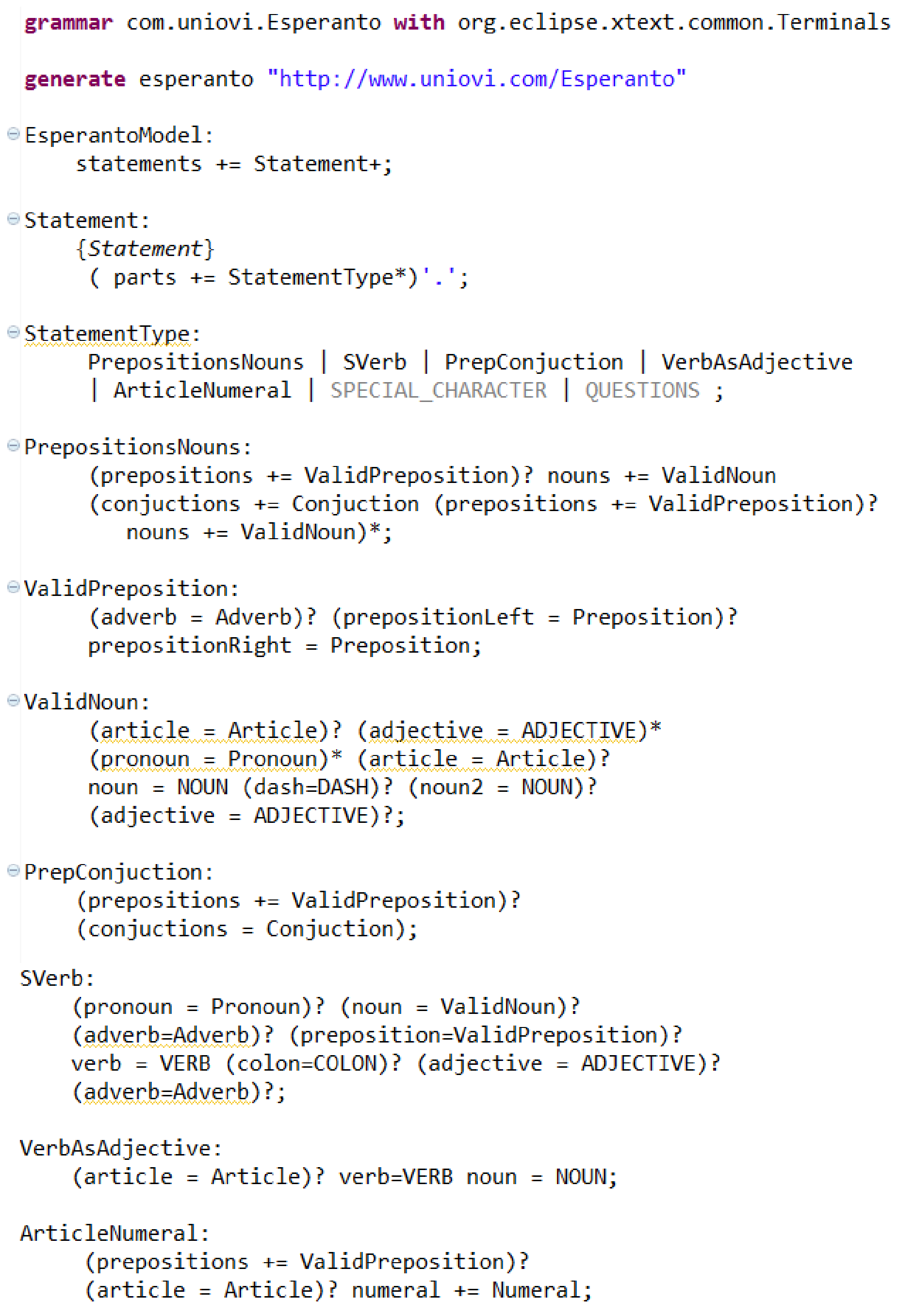


Figure 6.1: Metamodel that implements the Esperanto grammar.

The model created has been represented in figure 6.1. EsperantoModel contains a series of Statements, where each Statement is composed of different parts, which are of type StatementType. A StatementType can be of type PrepositionsNouns, Sverb, PrepConjuction, VerbAsAdjective, ArticleNumeral, SpecialCharacter or Questions and so on. This model contains the behaviour of the most common type of structures in the Esperanto language.

We use Xtext's grammar to define our Esperanto DSL (see figure 6.2,figure 6.3, figure 6.4).



```
grammar com.uniovi.Esperanto with org.eclipse.xtext.common.Terminals

generate esperanto "http://www.uniovi.com/Esperanto"

EsperantoModel:
    statements += Statement+;

Statement:
    {Statement}
     ( parts += StatementType*)'.';

StatementType:
     PrepositionsNouns | SVerb | PrepConjuction | VerbAsAdjective
     | ArticleNumeral | SPECIAL_CHARACTER | QUESTIONS ;

PrepositionsNouns:
     (prepositions += ValidPreposition)? nouns += ValidNoun
     (conjuctions += Conjuction (prepositions += ValidPreposition)?
        nouns += ValidNoun)*;

ValidPreposition:
     (adverb = Adverb)? (prepositionLeft = Preposition)?
     prepositionRight = Preposition;

ValidNoun:
     (article = Article)? (adjective = ADJECTIVE)*
     (pronoun = Pronoun)* (article = Article)?
     noun = NOUN (dash=DASH)? (noun2 = NOUN)?
     (adjective = ADJECTIVE)?;

PrepConjuction:
    (prepositions += ValidPreposition)?
    (conjuctions = Conjuction);

SVerb:
    (pronoun = Pronoun)? (noun = ValidNoun)?
    (adverb=Adverb)? (preposition=ValidPreposition)?
    verb = VERB (colon=COLON)? (adjective = ADJECTIVE)?
    (adverb=Adverb)?;

VerbAsAdjective:
    (article = Article)? verb=VERB noun = NOUN;

ArticleNumeral:
     (prepositions += ValidPreposition)?
     (article = Article)? numeral += Numeral;
```

Figure 6.2: Definition of the main structures in Xtext

In figure 6.2 is shown the main definition of the grammar. We start by creating the EsperantoModel, which will contain a list of Statements.Each statement will have a list named parts which will contain several Statement-Types.StatementType can be a PrepositionsNouns, Sverb, PrepConjuction, VerbAsAdjective, ArticleNumeral, Special character or Questions. In order to understand why the different structures were included in the model, we provide some examples of the kind of sentences that each structure would contain (see table 6.1) .

Structure	**Esperanto**	**English**
PrepositionsNouns	por la honoro kaj da gloroj.	for honor and glory
ValidPreposition	preter nia auto	past our car
ValidNoun	la bela grandega strando domo	the beautiful huge beach house
PrepConjuction	trans kaj	across and
SVerb	lia auto preskau kraso bela malbone hierau	his car almost crash pretty badly yesterday
VerbAsAdjective	la kuranta auto.	the running car
ArticleNumeral	dudek du mil sepcent dudek nau	twenty two thousands, seven hundreds, two tens and nine

Table 6.1: Examples of the main structures in our model

We believe its important to note that the order of the members in the syntax is relevant.

PrepositionsNouns has the next structure:

- An optional list of prepositions
- A list of nouns
- A list of conjunctions
- An optional second list of prepositions
- Multiple optional lists of prepositions

ValidPrepositions has the following members:

- An optional adverb
- An optional preposition
- A preposition

ValidNoun has the following structure:

- An optional article
- Multiple optional adjectives
- Multiple optional pronouns
- An optional second article
- A noun
- An optional dash
- An optional second noun
- An optional second adjective

PrepConjuction has the next structure:

- An optional list of prepositions
- A conjunction

Sverb has the following members:

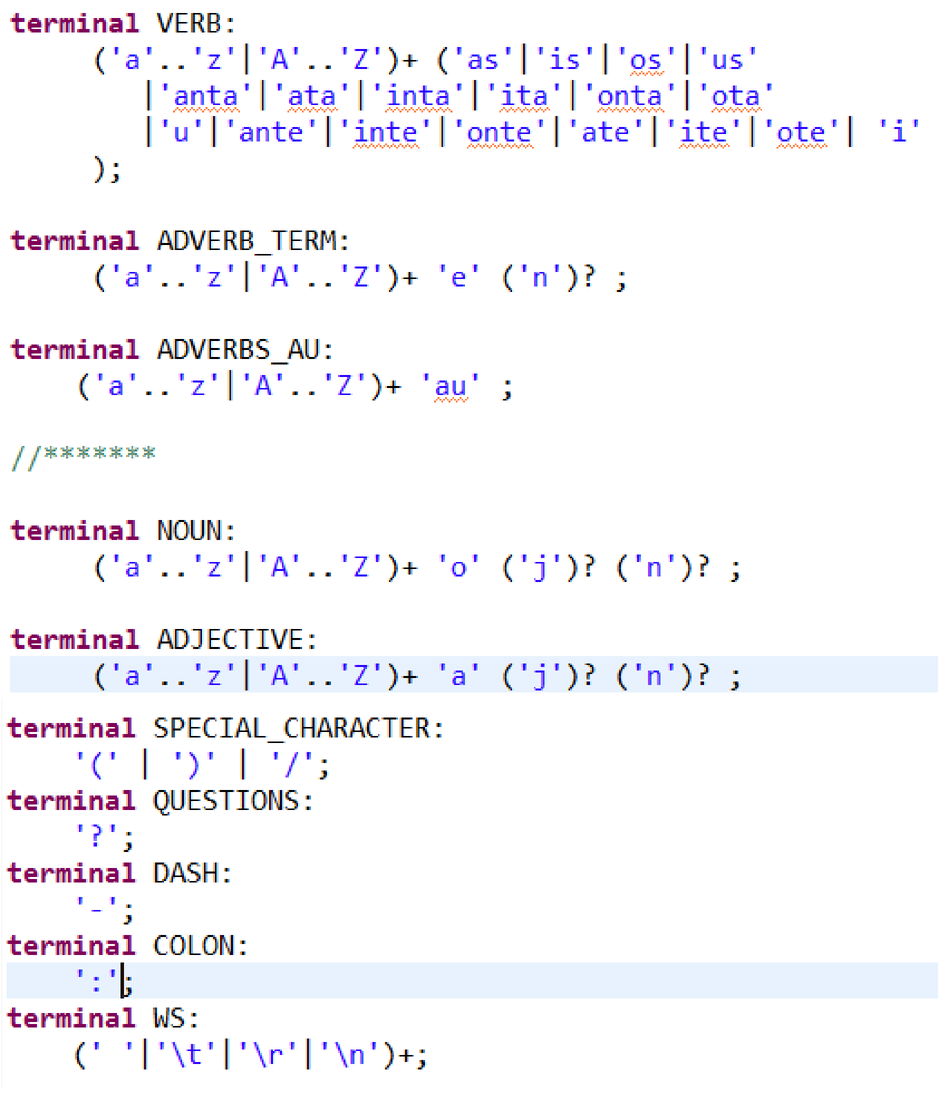
- An optional pronoun
- An optional noun
- An optional adverb
- An optional preposition
- A verb
- An optional colon
- An optional adjective
- An optional second adverb

VerbAsAdjective has the following structure:

- An optional article
- A verb
- A noun

ArticleNumeral has the next structure:

- An optional list of prepositions
- An optional article
- A list of numerals

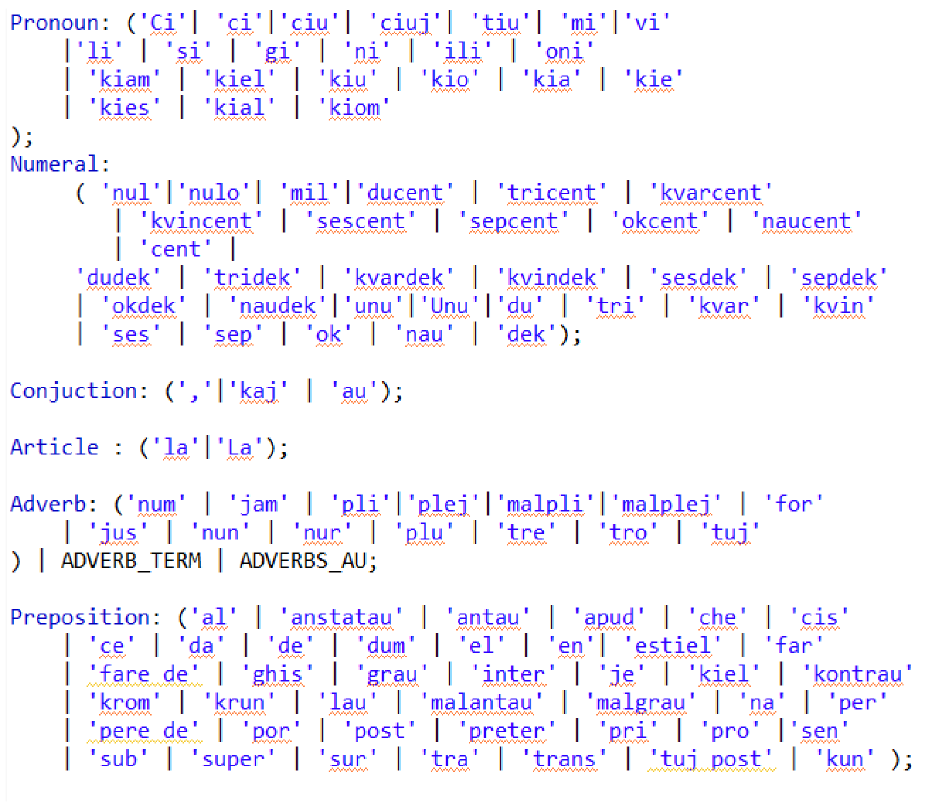


```
terminal VERB:
     ('a'..'z'|'A'..'Z')+ ('as'|'is'|'os'|'us'
        |'anta'|'ata'|'inta'|'ita'|'onta'|'ota'
        |'u'|'ante'|'inte'|'onte'|'ate'|'ite'|'ote'| 'i'
     );

terminal ADVERB_TERM:
     ('a'..'z'|'A'..'Z')+ 'e' ('n')? ;

terminal ADVERBS_AU:
    ('a'..'z'|'A'..'Z')+ 'au' ;

//*******

terminal NOUN:
     ('a'..'z'|'A'..'Z')+ 'o' ('j')? ('n')? ;

terminal ADJECTIVE:
     ('a'..'z'|'A'..'Z')+ 'a' ('j')? ('n')? ;

terminal SPECIAL_CHARACTER:
    '(' | ')' | '/';
terminal QUESTIONS:
    '?';
terminal DASH:
    '-';
terminal COLON:
    ':';
terminal WS:
    (' '|'\t'|'\r'|'\n')+;
```

Figure 6.3: Terminal rules definition in Xtext

The figure 6.3 contains the terminal rules used to identify different types of words, such as verbs, adjectives,etc.

- The verb identification rule categorizes as verbs words ending in “as, is, us, anta, ata, inta, ita, onta, ota, u, ante, inte, onte, ate, ite, ote, i”.
- We use two rules to identify adverbs, one that captures words ending in “e” or “en”, and another for words ending in “au”.
- The noun rule categorizes as nouns words ending in “o” “oj” or “ojn”.
- The adjective rule categorizes as adjectives words ending in “a” “aj” or “ajn”.
- We use many auxiliary rules to capture special characters, such as “(“, “)” and “/”, questions, “?”, dashes, “-”, colons “:” and a rule to identify end of line (WS rule).



```
Pronoun: ('Ci'| 'ci'|'ciu'| 'ciuj'| 'tiu'| 'mi'|'vi'
    |'li' | 'si' | 'gi' | 'ni' | 'ili' | 'oni'
    | 'kiam' | 'kiel' | 'kiu' | 'kio' | 'kia' | 'kie'
    | 'kies' | 'kial' | 'kiom'
);
Numeral:
     ( 'nul'|'nulo'| 'mil'|'ducent' | 'tricent' | 'kvarcent'
       | 'kvincent' | 'sescent' | 'sepcent' | 'okcent' | 'naucent'
       | 'cent' |
     'dudek' | 'tridek' | 'kvardek' | 'kvindek' | 'sesdek' | 'sepdek'
     | 'okdek' | 'naudek'|'unu'|'Unu'|'du' | 'tri' | 'kvar' | 'kvin'
     | 'ses' | 'sep' | 'ok' | 'nau' | 'dek');

Conjuction: (','|'kaj' | 'au');

Article : ('la'|'La');

Adverb: ('num' | 'jam' | 'pli'|'plej'|'malpli'|'malplej' | 'for'
    | 'jus' | 'nun' | 'nur' | 'plu' | 'tre' | 'tro' | 'tuj'
) | ADVERB_TERM | ADVERBS_AU;

Preposition: ('al' | 'anstatau' | 'antau' | 'apud' | 'che' | 'cis'
    | 'ce' | 'da' | 'de' | 'dum' | 'el' | 'en'| 'estiel' | 'far'
    | 'fare de' | 'ghis' | 'grau' | 'inter' | 'je' | 'kiel' | 'kontrau'
    | 'krom' | 'krun' | 'lau' | 'malantau' | 'malgrau' | 'na' | 'per'
    | 'pere de' | 'por' | 'post' | 'preter' | 'pri' | 'pro' |'sen'
    | 'sub' | 'super' | 'sur' | 'tra' | 'trans' | 'tuj post' | 'kun' );
```

Figure 6.4: Parser rules definition in Xtext

The figure 6.4 contains different parser rules to identify pronouns, numerals, conjunctions, articles, adverbs and prepositions.

- We use a list of the most common pronouns in Esperanto.
- We store a list of the most common prefixes for numerals in Esperanto.
- We use a comma ",", and ("kaj" and "au" as conjunctions.
- The only definite article in esperanto is "la", alike for all genders, cases and numbers.
- We use prefixes for the formation of adverbs.
- We store a list of the most common prepositions in Esperanto.

7. Extraction of UML Class Diagram

In this section, we will present the different heuristic rules employed to extract the class diagram. These rules were presented in Ibrahim and Ahmad [2010].

7.1 Concept Identification

The first step needed to obtain a class diagram from natural language requirements is to identify the concepts present in it. To identify and extract concepts we will:

1. Identify the stop words. In our case, we found that typically these are usually prepositions, articles, pronouns and conjuctions.
2. Calculate the total number of words T in the document without the stop words, the number of occurrences O_w of each word, and then calculate the frequency F of each word, according to:

$$F = O_w/T \tag{7.1}$$

3. Save the identified proper nouns, noun phrases and verbs as concepts.

7.2 Class Identification

After extracting the concepts, we can process them to extract candidate classes by applying the following rules:

1. If a concept only appears once in the document and its frequency is less than 2%, then it will be ignored.
2. If a concept is related to the design elements, then it will be ignored.
3. If a concept is related to Location name or People name, then it will be ignored.
4. If a concept is an attribute, then it will be ignored.

5. If a concept is a noun phrase (noun + noun) and the secound noun is an attribute, then the first noun is a class and the second one is an attribute of that class.
6. If a concept does not satisfy any of the rules, then it's likely a class.

7.3 Attribute Identification

The following rules assisted us in extracting attributes:

1. If a class is followed by the verb "have/has" and a colon, then the following enumeration is a list of attributes of the class.
2. If a concept is a noun phrase (noun + noun) and the secound noun is an attribute, then the first noun is a class and the second one is an attribute of that class.

7.4 Relationship Identification

Performing verb analysis and applying the next rules we have extracted relationships:

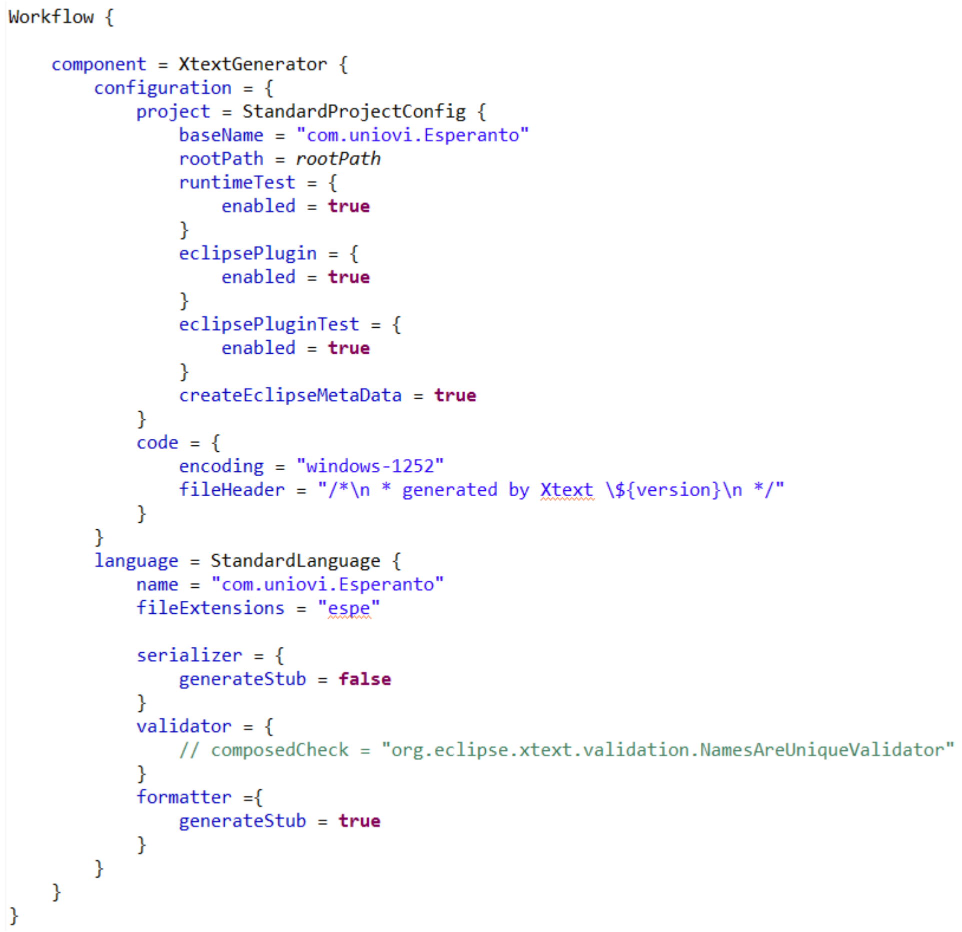
1. If we have a compound name (noun + noun) and the second one is a class, then the compound name is a generalization to the already identified class (second noun).
2. If the concept is a verb, and we can find a sentence in the from C1-V-C2, where C1 and C2 are classes, then V is an Association relationship.
3. If the concept is a verb, satisfies Relationship identification Rule 2 and the verb is one of the following (in Esperanto) ["require", "depends on", "rely on", "based on", "uses", "follows"] then the relationship discovered is a Dependency relationship.
4. Given a sentence in the form C1 + R1 + C2 + AND + C3 where C1, C2 and C3 are classes and R1 is a relationship, then the relationship R1 applies between the classes C1 and C2 and between C1 and C3.

8. Implementation

In this section, we will show the role of the technologies employed in our system. As previously stated, Xtext has been the tool chosen to develop our system. Xtext is used to create the DSL that contains the grammar of the Esperanto language. When creating a new Xtext project, the following projects are created:

- A project that contains the grammar definition and all language-specific components (parser,lexer,linker,validation,etc.)
- A project to run unit tests for the language
- A project to manage platform-independent IDE functionality (e.g. services for content assist)
- A project that contains the eclipse editor and other workbench related functionality
- A project to perform unit test for the editor.

We use Xtext's grammar to define our Esperanto DSL (see section §6). We also have a "mwe2" extension file(figure 8.1)where we can customize several options about our project, such as if we want to generate a serializer, a validator, the extension files for our language, and so on.

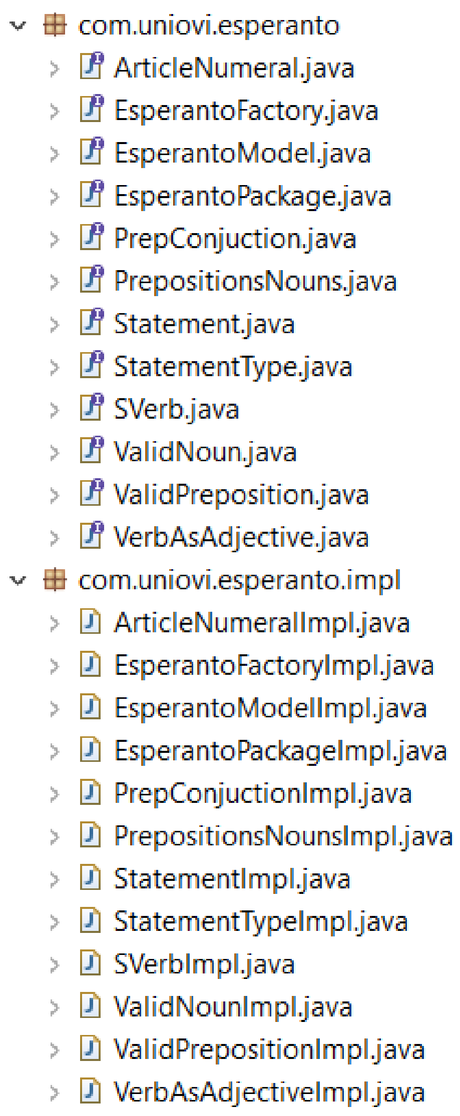


```
Workflow {

    component = XtextGenerator {
        configuration = {
            project = StandardProjectConfig {
                baseName = "com.uniovi.Esperanto"
                rootPath = rootPath
                runtimeTest = {
                    enabled = true
                }
                eclipsePlugin = {
                    enabled = true
                }
                eclipsePluginTest = {
                    enabled = true
                }
                createEclipseMetaData = true
            }
            code = {
                encoding = "windows-1252"
                fileHeader = "/*\n * generated by Xtext \${version}\n */"
            }
        }
        language = StandardLanguage {
            name = "com.uniovi.Esperanto"
            fileExtensions = "espe"

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

Figure 8.1: Configuration file

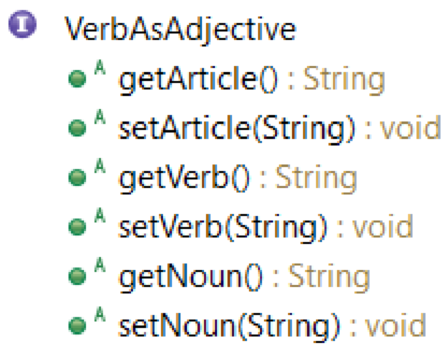
We would like to point out in figure 8.1 the “ fileExtensions ” attribute which will dictate the extension of the files that the users will have to use, as we will explain later.

One of the more helpful features of Xtext is the maping to objects from the elements in the syntax definition, Xtext creates Java objects for the elements used in the grammar (see figure 8.2) . The package “com.uniovi.esperanto” contains interfaces that are implemented by the classes pressent in the package “com.uniovi.esperanto.impl”.



```
com.uniovi.esperanto
    ArticleNumeral.java
    EsperantoFactory.java
    EsperantoModel.java
    EsperantoPackage.java
    PrepConjuction.java
    PrepositionsNouns.java
    Statement.java
    StatementType.java
    SVerb.java
    ValidNoun.java
    ValidPreposition.java
    VerbAsAdjective.java
com.uniovi.esperanto.impl
    ArticleNumeralImpl.java
    EsperantoFactoryImpl.java
    EsperantoModelImpl.java
    EsperantoPackageImpl.java
    PrepConjuctionImpl.java
    PrepositionsNounsImpl.java
    StatementImpl.java
    StatementTypeImpl.java
    SVerbImpl.java
    ValidNounImpl.java
    ValidPrepositionImpl.java
    VerbAsAdjectiveImpl.java
```

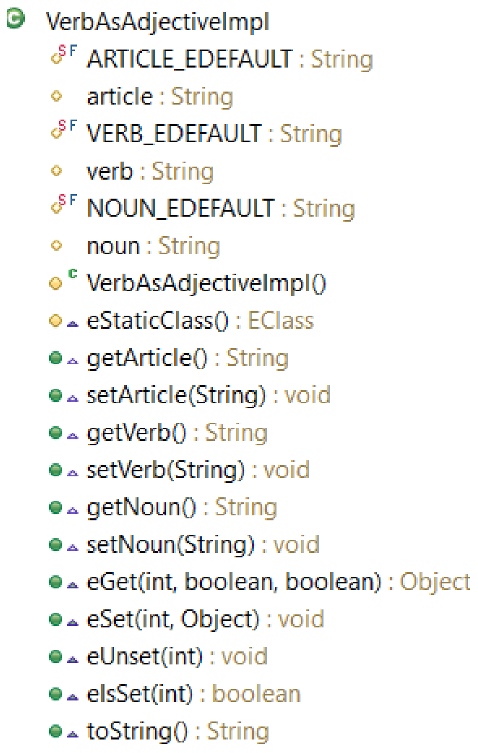
Figure 8.2: Mapping to Objects



```
VerbAsAdjective
    getArticle() : String
    setArticle(String) : void
    getVerb() : String
    setVerb(String) : void
    getNoun() : String
    setNoun(String) : void
```

Figure 8.3: Interface of VerbAsAdjective

For example, in figure 8.3 and figure 8.4 we can see the mapping made for "VerbAsAdjective", containing an interface and a class, one of the simplest structures in our grammar.

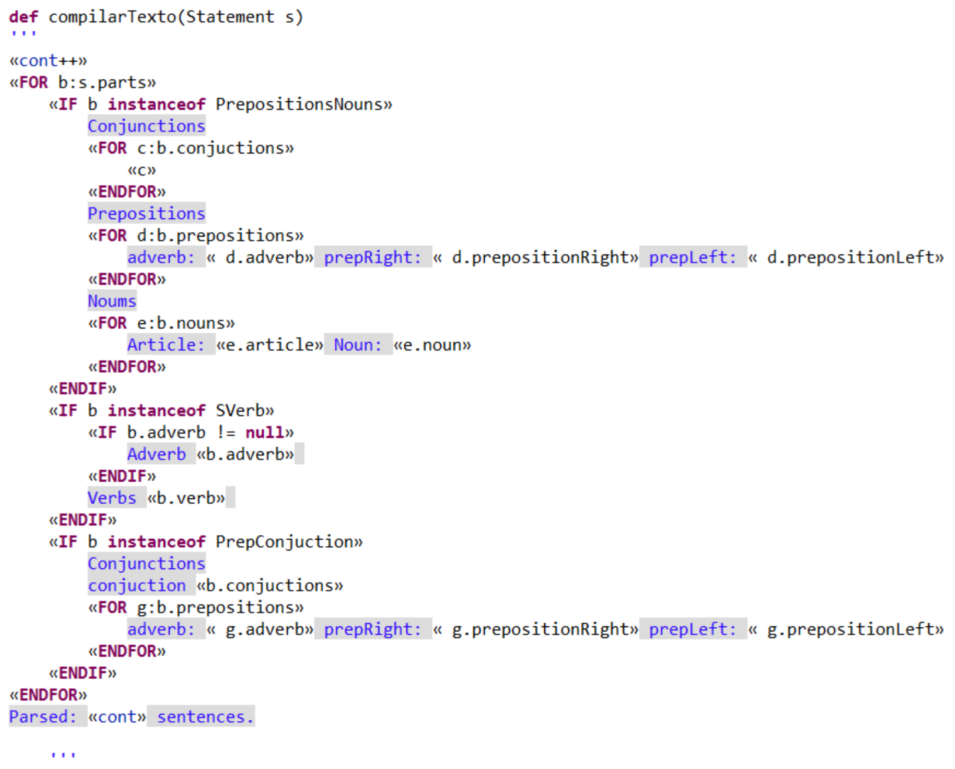


```
VerbAsAdjectiveImpl
    ARTICLE_EDEFAULT : String
    article : String
    VERB_EDEFAULT : String
    verb : String
    NOUN_EDEFAULT : String
    noun : String
    VerbAsAdjectiveImpl()
    eStaticClass() : EClass
    getArticle() : String
    setArticle(String) : void
    getVerb() : String
    setVerb(String) : void
    getNoun() : String
    setNoun(String) : void
    eGet(int, boolean, boolean) : Object
    eSet(int, Object) : void
    eUnset(int) : void
    eIsSet(int) : boolean
    toString() : String
```

Figure 8.4: Implementation of VerbAsAdjective

In our case, we used two generators, one to generate a text file with the processed output of a user's requirements and another to generate the UML Class diagram. We have used Java and Xtend for these generators. The behaviour of Xtext and its generators is that once the user saves the changes made in his editor, the generators are automatically executed, generating the different outputs when the user saves.

The Code 8.5 contains the code used for the text generator, in this case, we used a Xpand template. The text generator was not developed to contain all the structures in the grammar, as it was used on the early stages of development to better understand Xtext's behavior.

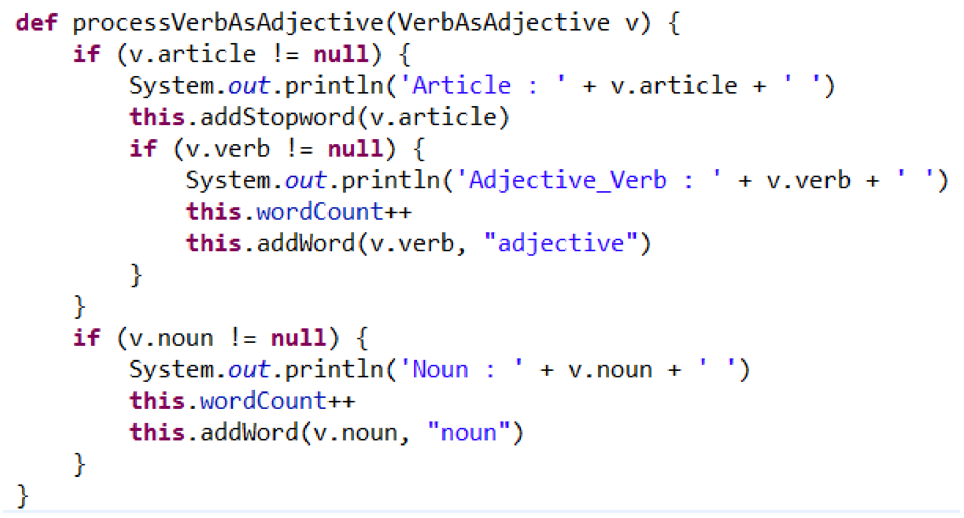


```
def compilarTexto(Statement s)
'''
«cont++»
«FOR b:s.parts»
    «IF b instanceof PrepositionsNouns»
        Conjunctions
        «FOR c:b.conjuctions»
            «c»
        «ENDFOR»
        Prepositions
        «FOR d:b.prepositions»
            adverb: « d.adverb» prepRight: « d.prepositionRight» prepLeft: « d.prepositionLeft»
        «ENDFOR»
        Noums
        «FOR e:b.nouns»
            Article: «e.article» Noun: «e.noun»
        «ENDFOR»
    «ENDIF»
    «IF b instanceof SVerb»
        «IF b.adverb != null»
            Adverb «b.adverb»
        «ENDIF»
        Verbs «b.verb»
    «ENDIF»
    «IF b instanceof PrepConjuction»
        Conjunctions
        conjuction «b.conjuctions»
        «FOR g:b.prepositions»
            adverb: « g.adverb» prepRight: « g.prepositionRight» prepLeft: « g.prepositionLeft»
        «ENDFOR»
    «ENDIF»
«ENDFOR»
Parsed: «cont» sentences.

    '''
```

Figure 8.5: Text generator

The generator used to create the UML diagram is a long program, with over 900 lines of code, we will simply show the most interesting parts.

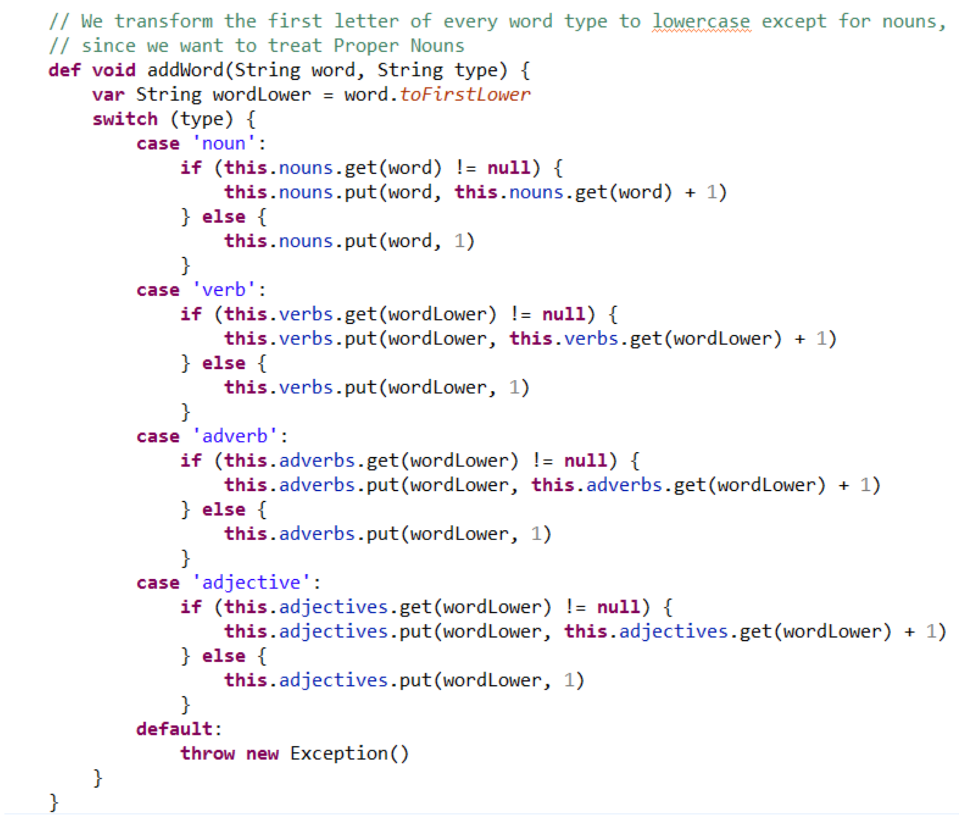
In Code 8.6 is the processing of VerbAsAdjective, in this case, we take into consideration the present verbs and nouns to the word count, and we store the words in their respective lists, according to their type.



```
def processVerbAsAdjective(VerbAsAdjective v) {
    if (v.article != null) {
        System.out.println('Article : ' + v.article + ' ')
        this.addStopword(v.article)
        if (v.verb != null) {
            System.out.println('Adjective_Verb : ' + v.verb + ' ')
            this.wordCount++
            this.addWord(v.verb, "adjective")
        }
    }
    if (v.noun != null) {
        System.out.println('Noun : ' + v.noun + ' ')
        this.wordCount++
        this.addWord(v.noun, "noun")
    }
}
```

Figure 8.6: Initial processing of VerbAsAdjective

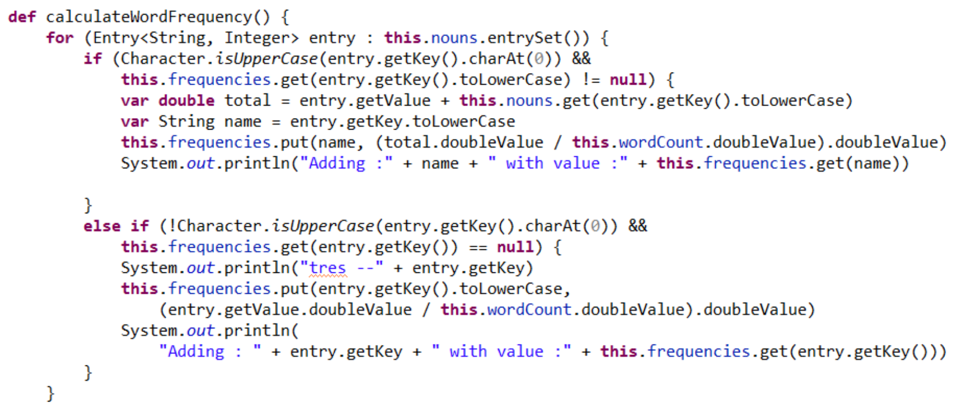
In Code 8.7 is the method used to add the different type of words.



```
// We transform the first letter of every word type to lowercase except for nouns,
// since we want to treat Proper Nouns
def void addWord(String word, String type) {
    var String wordLower = word.toFirstLower
    switch (type) {
        case 'noun':
            if (this.nouns.get(word) != null) {
                this.nouns.put(word, this.nouns.get(word) + 1)
            } else {
                this.nouns.put(word, 1)
            }
        case 'verb':
            if (this.verbs.get(wordLower) != null) {
                this.verbs.put(wordLower, this.verbs.get(wordLower) + 1)
            } else {
                this.verbs.put(wordLower, 1)
            }
        case 'adverb':
            if (this.adverbs.get(wordLower) != null) {
                this.adverbs.put(wordLower, this.adverbs.get(wordLower) + 1)
            } else {
                this.adverbs.put(wordLower, 1)
            }
        case 'adjective':
            if (this.adjectives.get(wordLower) != null) {
                this.adjectives.put(wordLower, this.adjectives.get(wordLower) + 1)
            } else {
                this.adjectives.put(wordLower, 1)
            }
        default:
            throw new Exception()
    }
}
```

Figure 8.7: Method used to include words in their category list

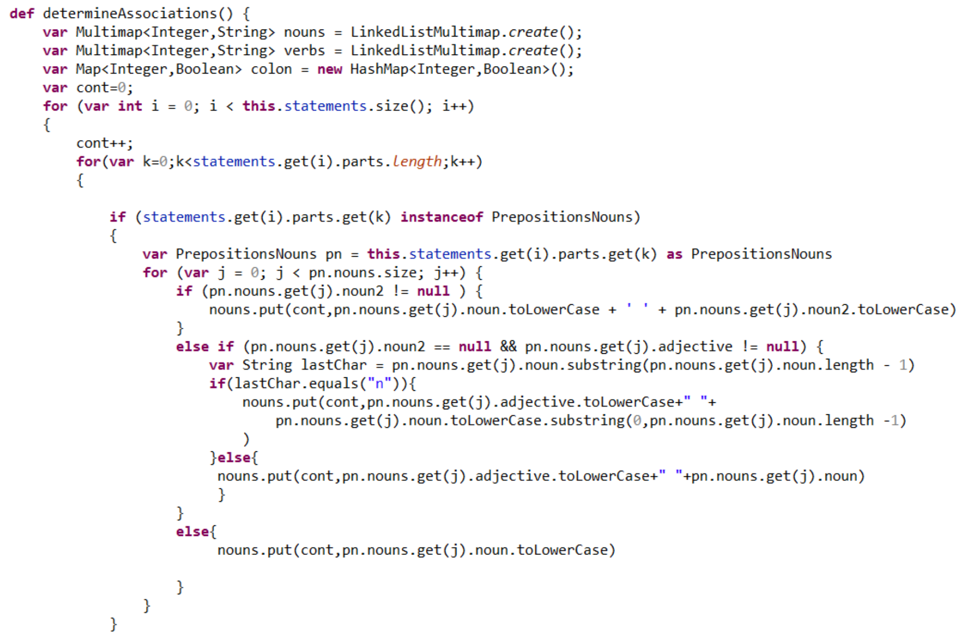
In Code 8.8 we can see a fragment of code to calculate the word frequency. In this case, we have to check if the noun has a capital starting letter.



```
def calculateWordFrequency() {
    for (Entry<String, Integer> entry : this.nouns.entrySet()) {
        if (Character.isUpperCase(entry.getKey().charAt(0)) &&
            this.frequencies.get(entry.getKey().toLowerCase) != null) {
            var double total = entry.getValue + this.nouns.get(entry.getKey().toLowerCase)
            var String name = entry.getKey.toLowerCase
            this.frequencies.put(name, (total.doubleValue / this.wordCount.doubleValue).doubleValue)
            System.out.println("Adding :" + name + " with value :" + this.frequencies.get(name))

        }
        else if (!Character.isUpperCase(entry.getKey().charAt(0)) &&
            this.frequencies.get(entry.getKey()) == null) {
            System.out.println("tres --" + entry.getKey)
            this.frequencies.put(entry.getKey().toLowerCase,
                (entry.getValue.doubleValue / this.wordCount.doubleValue).doubleValue)
            System.out.println(
                "Adding : " + entry.getKey + " with value :" + this.frequencies.get(entry.getKey()))
        }
    }
```

Figure 8.8: Fragment of code to obtain the word frequency.

Another part of the UML generator that we would like to show is in Code 8.9 , the association analysis is a complex task, in the fragment shown we can observe how we iterate through the statements, with different approaches according to the type of the words.

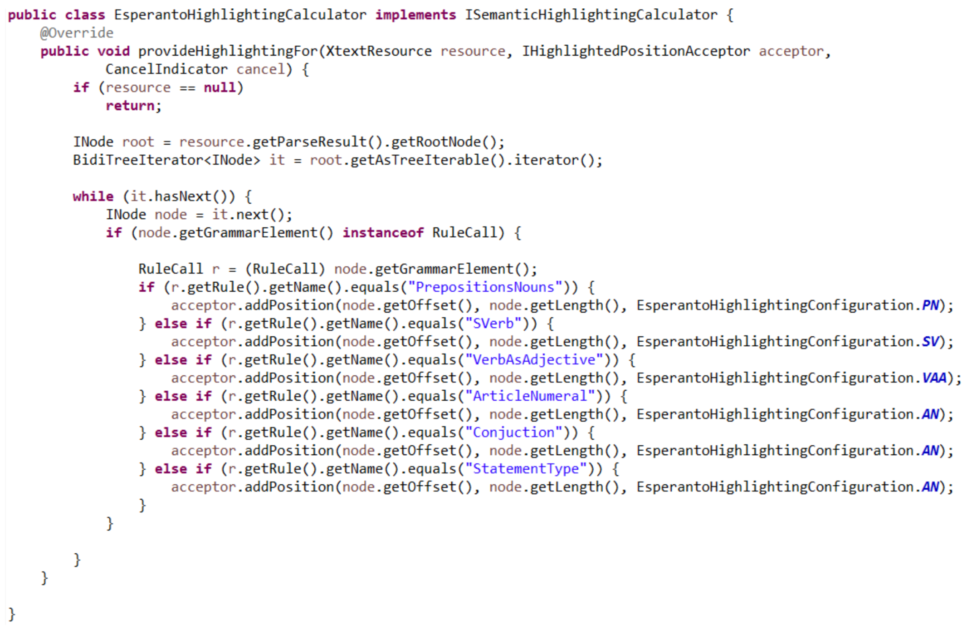


```
def determineAssociations() {
    var Multimap<Integer,String> nouns = LinkedListMultimap.create();
    var Multimap<Integer,String> verbs = LinkedListMultimap.create();
    var Map<Integer,Boolean> colon = new HashMap<Integer,Boolean>();
    var cont=0;
    for (var int i = 0; i < this.statements.size(); i++)
    {
        cont++;
        for(var k=0;k<statements.get(i).parts.length;k++)
        {

            if (statements.get(i).parts.get(k) instanceof PrepositionsNouns)
            {
                var PrepositionsNouns pn = this.statements.get(i).parts.get(k) as PrepositionsNouns
                for (var j = 0; j < pn.nouns.size; j++) {
                    if (pn.nouns.get(j).noun2 != null ) {
                        nouns.put(cont,pn.nouns.get(j).noun.toLowerCase + ' ' + pn.nouns.get(j).noun2.toLowerCase)
                    }
                    else if (pn.nouns.get(j).noun2 == null && pn.nouns.get(j).adjective != null) {
                        var String lastChar = pn.nouns.get(j).noun.substring(pn.nouns.get(j).noun.length - 1)
                        if(lastChar.equals("n")){
                            nouns.put(cont,pn.nouns.get(j).adjective.toLowerCase+" "+
                                pn.nouns.get(j).noun.toLowerCase.substring(0,pn.nouns.get(j).noun.length -1)
                            )
                        }else{
                         nouns.put(cont,pn.nouns.get(j).adjective.toLowerCase+" "+pn.nouns.get(j).noun)
                         }
                    }
                    else{
                         nouns.put(cont,pn.nouns.get(j).noun.toLowerCase)

                    }
                }
            }
```

Figure 8.9: Fragment of code for association determination

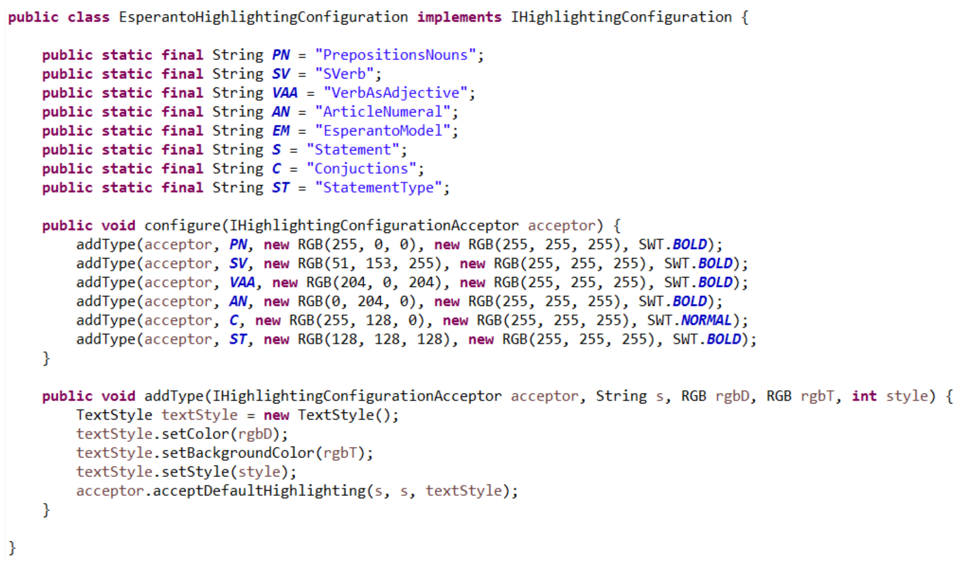
We also customized the behavior of the editor's syntax highlighting and the outline view, for the latter, we displayed the simple elements that compose the grammatical structures.

The highlighting in the editor is obtained through the following code (Code 8.10 and Code 8.11) :



```
public class EsperantoHighlightingCalculator implements ISemanticHighlightingCalculator {
    @Override
    public void provideHighlightingFor(XtextResource resource, IHighlightedPositionAcceptor acceptor,
            CancelIndicator cancel) {
        if (resource == null)
            return;

        INode root = resource.getParseResult().getRootNode();
        BidiTreeIterator<INode> it = root.getAsTreeIterable().iterator();

        while (it.hasNext()) {
            INode node = it.next();
            if (node.getGrammarElement() instanceof RuleCall) {

                RuleCall r = (RuleCall) node.getGrammarElement();
                if (r.getRule().getName().equals("PrepositionsNouns")) {
                    acceptor.addPosition(node.getOffset(), node.getLength(), EsperantoHighlightingConfiguration.PN);
                } else if (r.getRule().getName().equals("SVerb")) {
                    acceptor.addPosition(node.getOffset(), node.getLength(), EsperantoHighlightingConfiguration.SV);
                } else if (r.getRule().getName().equals("VerbAsAdjective")) {
                    acceptor.addPosition(node.getOffset(), node.getLength(), EsperantoHighlightingConfiguration.VAA);
                } else if (r.getRule().getName().equals("ArticleNumeral")) {
                    acceptor.addPosition(node.getOffset(), node.getLength(), EsperantoHighlightingConfiguration.AN);
                } else if (r.getRule().getName().equals("Conjuction")) {
                    acceptor.addPosition(node.getOffset(), node.getLength(), EsperantoHighlightingConfiguration.AN);
                } else if (r.getRule().getName().equals("StatementType")) {
                    acceptor.addPosition(node.getOffset(), node.getLength(), EsperantoHighlightingConfiguration.AN);
                }
            }

        }
    }

}
```

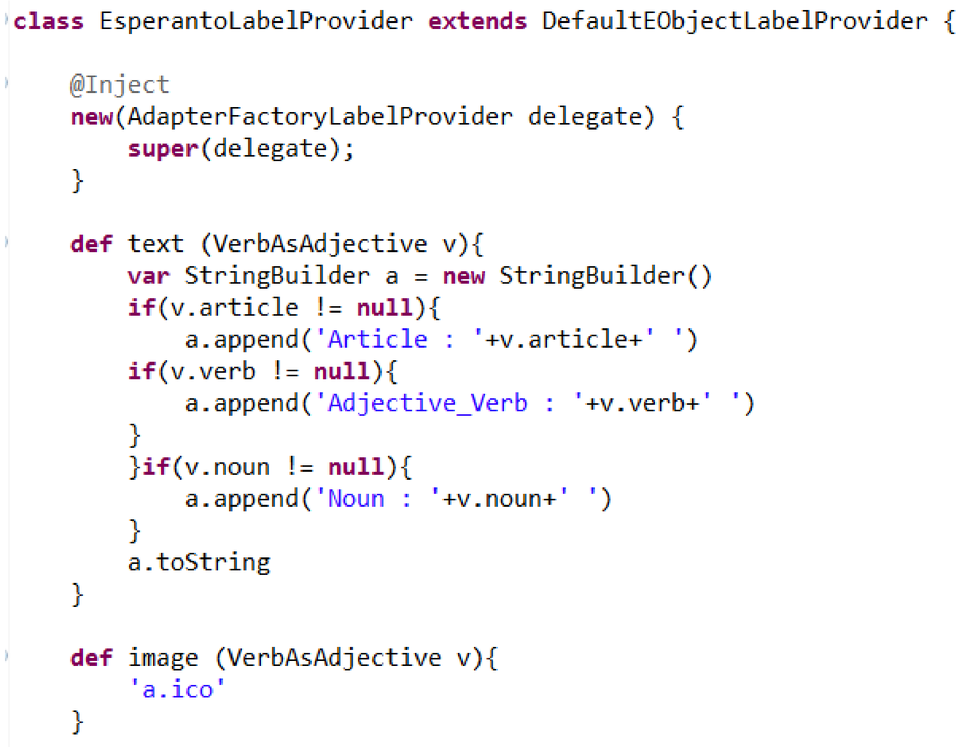
Figure 8.10: Code for the highlighting in the editor (I).



```java
public class EsperantoHighlightingConfiguration implements IHighlightingConfiguration {

    public static final String PN = "PrepositionsNouns";
    public static final String SV = "SVerb";
    public static final String VAA = "VerbAsAdjective";
    public static final String AN = "ArticleNumeral";
    public static final String EM = "EsperantoModel";
    public static final String S = "Statement";
    public static final String C = "Conjuctions";
    public static final String ST = "StatementType";

    public void configure(IHighlightingConfigurationAcceptor acceptor) {
        addType(acceptor, PN, new RGB(255, 0, 0), new RGB(255, 255, 255), SWT.BOLD);
        addType(acceptor, SV, new RGB(51, 153, 255), new RGB(255, 255, 255), SWT.BOLD);
        addType(acceptor, VAA, new RGB(204, 0, 204), new RGB(255, 255, 255), SWT.BOLD);
        addType(acceptor, AN, new RGB(0, 204, 0), new RGB(255, 255, 255), SWT.BOLD);
        addType(acceptor, C, new RGB(255, 128, 0), new RGB(255, 255, 255), SWT.NORMAL);
        addType(acceptor, ST, new RGB(128, 128, 128), new RGB(255, 255, 255), SWT.BOLD);
    }

    public void addType(IHighlightingConfigurationAcceptor acceptor, String s, RGB rgbD, RGB rgbT, int style) {
        TextStyle textStyle = new TextStyle();
        textStyle.setColor(rgbD);
        textStyle.setBackgroundColor(rgbT);
        textStyle.setStyle(style);
        acceptor.acceptDefaultHighlighting(s, s, textStyle);
    }

}
```

Figure 8.11: Code for the highlighting in the editor (II).

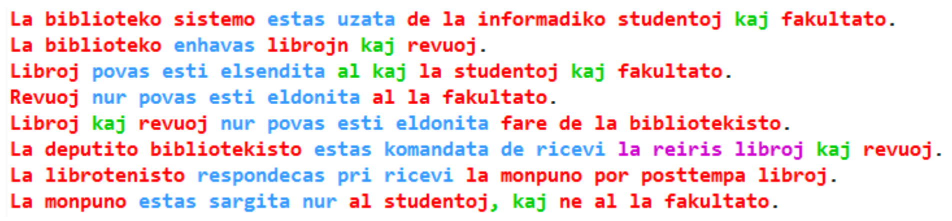
Some of the modifications done to the outline view are present in Code 8.12, in this particular case, we define the text and the images that should be shown in the outline.



```
class EsperantoLabelProvider extends DefaultEObjectLabelProvider {

    @Inject
    new(AdapterFactoryLabelProvider delegate) {
        super(delegate);
    }

    def text (VerbAsAdjective v){
        var StringBuilder a = new StringBuilder()
        if(v.article != null){
            a.append('Article : '+v.article+' ')
        if(v.verb != null){
            a.append('Adjective_Verb : '+v.verb+' ')
        }
        }if(v.noun != null){
            a.append('Noun : '+v.noun+' ')
        }
        a.toString
    }

    def image (VerbAsAdjective v){
        'a.ico'
    }
```

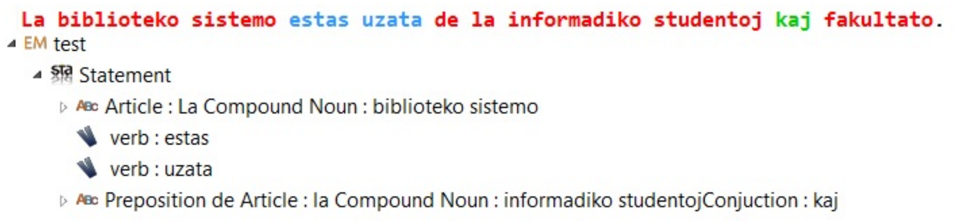
Figure 8.12: Fragment of code for the desired outline view of VerbAsAdjective elements

Code 8.13 and figure 8.14 illustrate the type of customization that can be achieved into the graphical editor, where each type of structure has its own color. In the outline, every member of each structure is categorized according to their type.



```
La biblioteko sistemo estas uzata de la informadiko studentoj kaj fakultato.
La biblioteko enhavas librojn kaj revuoj.
Libroj povas esti elsendita al kaj la studentoj kaj fakultato.
Revuoj nur povas esti eldonita al la fakultato.
Libroj kaj revuoj nur povas esti eldonita fare de la bibliotekisto.
La deputito bibliotekisto estas komandata de ricevi la reiris libroj kaj revuoj.
La librotenisto respondecas pri ricevi la monpuno por posttempa libroj.
La monpuno estas sargita nur al studentoj, kaj ne al la fakultato.
```

Figure 8.13: Sample of the customized editor view with some Esperanto text

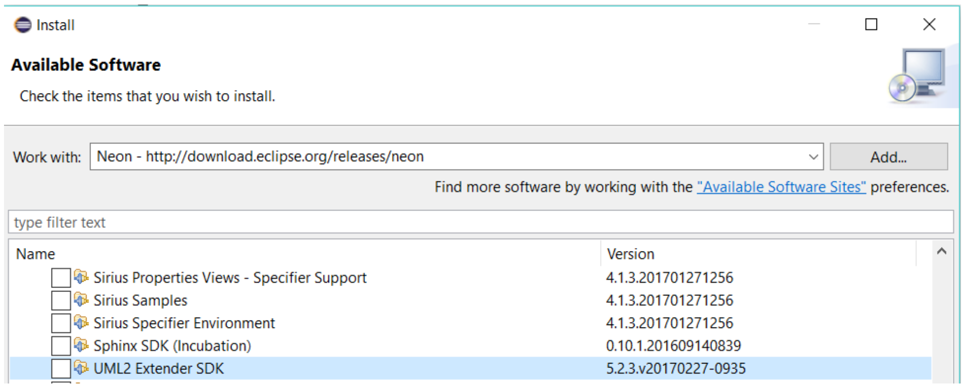


```
La biblioteko sistemo estas uzata de la informadiko studentoj kaj fakultato.
◢ EM test
  ◢ Sta Statement
    ▷ Article : La Compound Noun : biblioteko sistemo
      verb : estas
      verb : uzata
    ▷ Preposition de Article : la Compound Noun : informadiko studentojConjuction : kaj
```

Figure 8.14: Outline view customization

Furthermore, we used the UML2 package to create the diagram and its different components Kenn.Hussey [2011].

In order to use the UML2 package, we had to install the package, as shown in figure 8.15 and modify the file "MANIFEST.MF" inside the "META-INF" folder of the main project (see figure 8.16) :



Install

Available Software

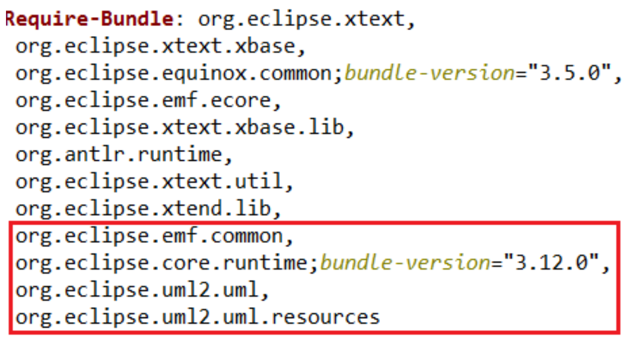
Check the items that you wish to install.

Work with: Neon - http://download.eclipse.org/releases/neon

Find more software by working with the "Available Software Sites" preferences.

Name	Version
☐ Sirius Properties Views - Specifier Support	4.1.3.201701271256
☐ Sirius Samples	4.1.3.201701271256
☐ Sirius Specifier Environment	4.1.3.201701271256
☐ Sphinx SDK (Incubation)	0.10.1.201609140839
☐ UML2 Extender SDK	5.2.3.v20170227-0935

Figure 8.15: UML2 package installation

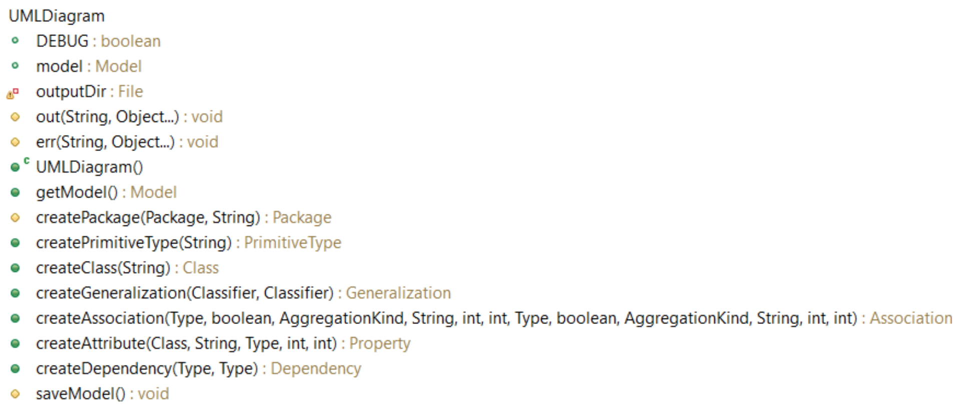


```
Require-Bundle: org.eclipse.xtext,
 org.eclipse.xtext.xbase,
 org.eclipse.equinox.common;bundle-version="3.5.0",
 org.eclipse.emf.ecore,
 org.eclipse.xtext.xbase.lib,
 org.antlr.runtime,
 org.eclipse.xtext.util,
 org.eclipse.xtend.lib,
 org.eclipse.emf.common,
 org.eclipse.core.runtime;bundle-version="3.12.0",
 org.eclipse.uml2.uml,
 org.eclipse.uml2.uml.resources
```

Figure 8.16: Dependencies added to manifest.mf to work with UML 2.0

Using the UML2.0 package, we created a class to create the different elements of a UML class diagram in a clean and straight way, covering the confusing notations and items that this package offers, shown in figure 8.17. We believe its a useful utility, that is why we have uploaded it to GitHub, so anyone can use it.

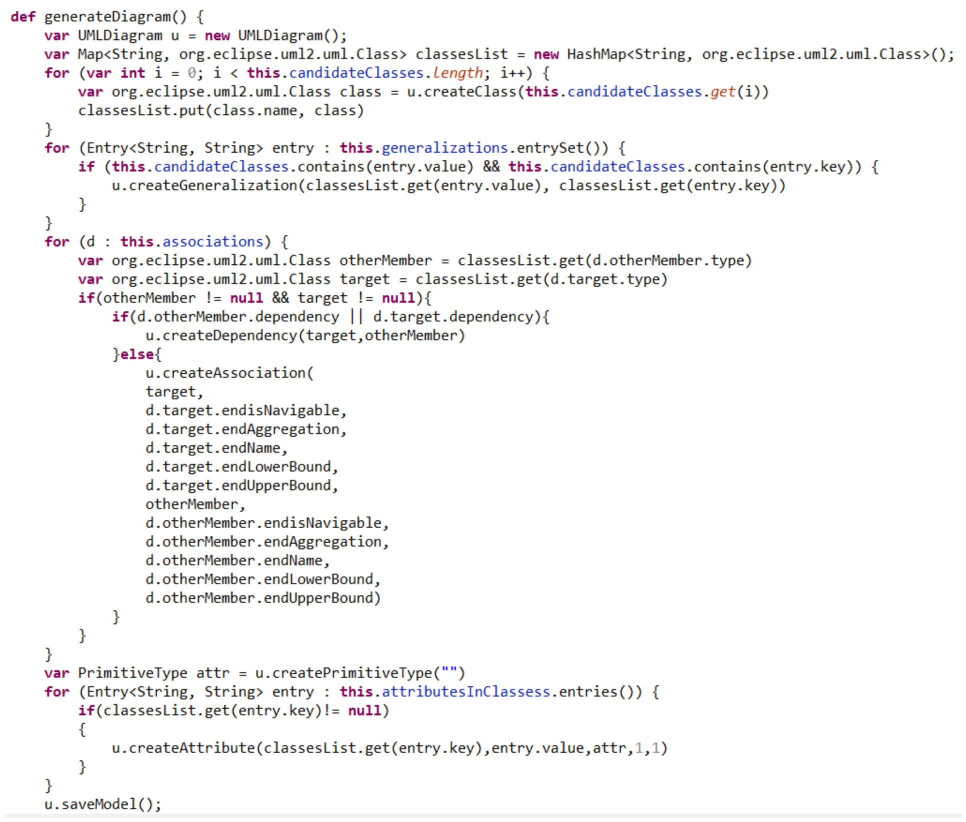
https://github.com/1bertillo/UML



```
UMLDiagram
  DEBUG : boolean
  model : Model
  outputDir : File
  out(String, Object...) : void
  err(String, Object...) : void
  UMLDiagram()
  getModel() : Model
  createPackage(Package, String) : Package
  createPrimitiveType(String) : PrimitiveType
  createClass(String) : Class
  createGeneralization(Classifier, Classifier) : Generalization
  createAssociation(Type, boolean, AggregationKind, String, int, int, Type, boolean, AggregationKind, String, int, int) : Association
  createAttribute(Class, String, Type, int, int) : Property
  createDependency(Type, Type) : Dependency
  saveModel() : void
```

Figure 8.17: Class to create a UML class diagram

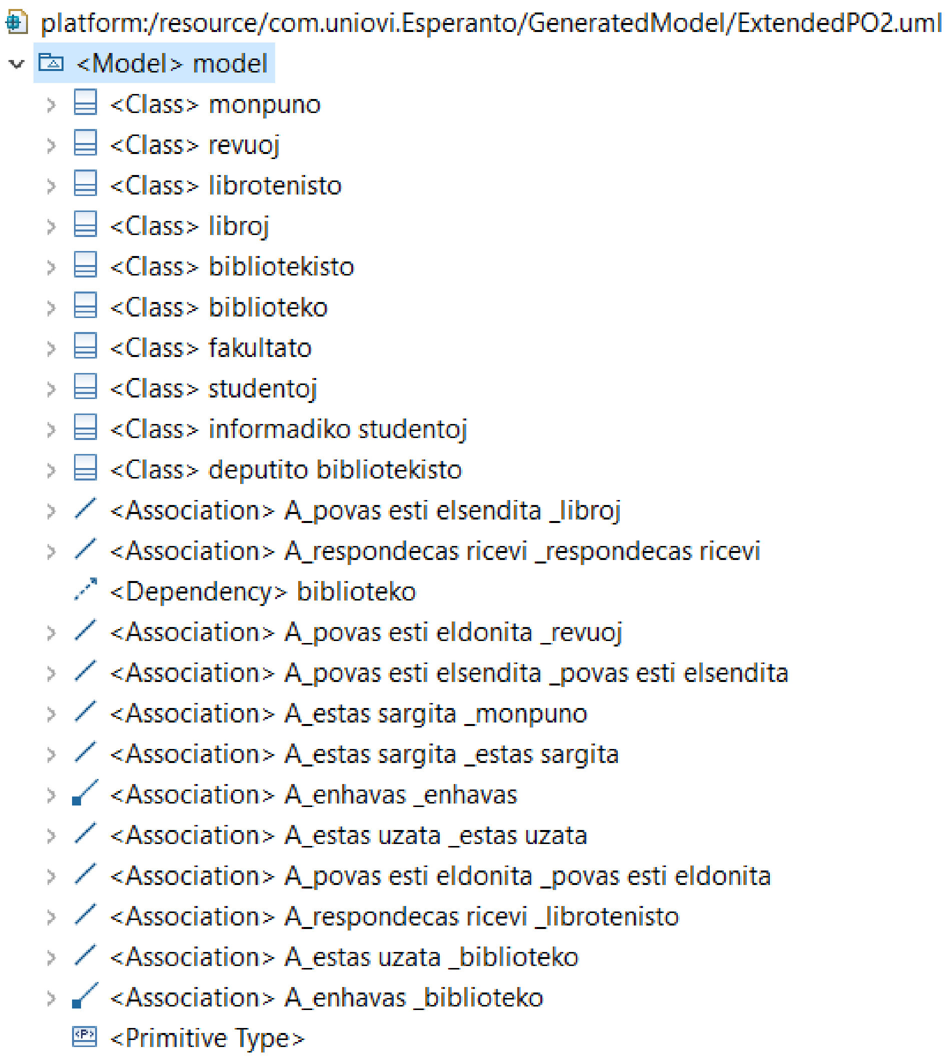
In our UML generator, in Code 8.18 is the code that creates the diagram when all the processing is completed.



```
def generateDiagram() {
    var UMLDiagram u = new UMLDiagram();
    var Map<String, org.eclipse.uml2.uml.Class> classesList = new HashMap<String, org.eclipse.uml2.uml.Class>();
    for (var int i = 0; i < this.candidateClasses.length; i++) {
        var org.eclipse.uml2.uml.Class class = u.createClass(this.candidateClasses.get(i))
        classesList.put(class.name, class)
    }
    for (Entry<String, String> entry : this.generalizations.entrySet()) {
        if (this.candidateClasses.contains(entry.value) && this.candidateClasses.contains(entry.key)) {
            u.createGeneralization(classesList.get(entry.value), classesList.get(entry.key))
        }
    }
    for (d : this.associations) {
        var org.eclipse.uml2.uml.Class otherMember = classesList.get(d.otherMember.type)
        var org.eclipse.uml2.uml.Class target = classesList.get(d.target.type)
        if(otherMember != null && target != null){
            if(d.otherMember.dependency || d.target.dependency){
                u.createDependency(target,otherMember)
            }else{
                u.createAssociation(
                target,
                d.target.endisNavigable,
                d.target.endAggregation,
                d.target.endName,
                d.target.endLowerBound,
                d.target.endUpperBound,
                otherMember,
                d.otherMember.endisNavigable,
                d.otherMember.endAggregation,
                d.otherMember.endName,
                d.otherMember.endLowerBound,
                d.otherMember.endUpperBound)
            }
        }
    }
    var PrimitiveType attr = u.createPrimitiveType("")
    for (Entry<String, String> entry : this.attributesInClassess.entries()) {
        if(classesList.get(entry.key)!= null)
        {
            u.createAttribute(classesList.get(entry.key),entry.value,attr,1,1)
        }
    }
    u.saveModel();
```

Figure 8.18: Code to generate the UML class diagram

The diagram generated is stored in a file with the extension “uml”, figure 8.19 contains a view of that file.



```
platform:/resource/com.uniovi.Esperanto/GeneratedModel/ExtendedPO2.uml
<Model> model
    <Class> monpuno
    <Class> revuoj
    <Class> librotenisto
    <Class> libroj
    <Class> bibliotekisto
    <Class> biblioteko
    <Class> fakultato
    <Class> studentoj
    <Class> informadiko studentoj
    <Class> deputito bibliotekisto
    <Association> A_povas esti elsendita _libroj
    <Association> A_respondecas ricevi _respondecas ricevi
    <Dependency> biblioteko
    <Association> A_povas esti eldonita _revuoj
    <Association> A_povas esti elsendita _povas esti elsendita
    <Association> A_estas sargita _monpuno
    <Association> A_estas sargita _estas sargita
    <Association> A_enhavas _enhavas
    <Association> A_estas uzata _estas uzata
    <Association> A_povas esti eldonita _povas esti eldonita
    <Association> A_respondecas ricevi _librotenisto
    <Association> A_estas uzata _biblioteko
    <Association> A_enhavas _biblioteko
    <Primitive Type>
```

Figure 8.19: UML model editor view of the generated class diagram

Furthermore, in order to generate a graphical representation of the diagram, we have used Papyrus Foundation [2007b], a graphical editing tool for UML2. In order to obtain a graphical representation, we have to use the file generated to create a new papyrus model, and select class diagram as its type. The graphical representation of the generated model can be seen in figure 9.1 .

9. Evaluation and discussion

We have used two methods to evaluate our system: we have compared our system against the approach proposed by Ibrahim and Ahmad [2010] and we have tested the diagram generated by our system against diagrams extracted by fellow engineers.

9.1 System vs system

In order to validate our system and obtain a general assessment for accuracy and efficiency of our system, we have tested it against the RACE tool developed by Ibrahim and Ahmad [2010]. We have used the same input requirements, a library system, with two changes:

1. We have added the following line (in Esperanto) to provide a dependency example: "The librarian depends on the library".
2. We have added attributes to the requirements. As we have not employed ontologies, only the attributes present in the text could be extracted into the diagram.

This is the original input used by RACE (English):\\ "The library System is used by the Informatics students and Faculty. The Library contains Books and Journals. Books can be issued to both the Students and Faculty. Journals can only be issued to the Faculty. Books and Journals can only be issued by the Librarian. The deputy-Librarian is in-charge of receiving the Returned Books and Journals. The Accountant is responsible for receiving the fine for over-due books. Fine is charged only to students, and not to the Faculty."\\ Whereas our input used for the testing (Esperanto) is:\\ "La biblioteko sistemo estas uzata de la informadiko studentoj kaj fakultato.La biblioteko enhavas librojn kaj revuoj.Libroj povas esti elsendita al kaj la studentoj kaj fakultato.Revuoj nur povas esti eldonita al la fakultato.Libroj kaj revuoj nur povas esti eldonita fare de la bibliotekisto.La deputito bibliotekisto estas komandata de ricevi la reiris libroj kaj revuoj.La librotenisto respondecas pri ricevi la monpuno por posttempa libroj.La monpuno estas sargita nur al studentoj, kaj ne al la fakultato. La bibliotekisto dependas de la biblioteko.Unu libroj havas: nomo, titolo, autoro, eldonisto, isbnno, identingo, genro kaj la publikigas dato.La biblioteko havas: nomo, malferma tempo,proksima tempo kaj legantoj.La studentoj havas: nomo, sekso kaj membreco nombro.La deputito bibliotekisto havas: nomo, sekso, salajro kaj membreco nombro.La fakultato havas: nomo kaj membreco nombro.Fakultato estas savita fakultato nomo, kaj membreco nombro.La monpuno havas: kvanto kaj la ricevi daton.Bibliotekisto estas konservita bibliotekisto nomo, bibliotekisto salajro, bibliotekisto sekso kaj bibliotekisto identingo.Unu revuoj havas: nomo, titolo, eldonisto, isbnno kaj la publikigas dato.Unu librotenisto havas: nomo, salajro, sekso kaj identingo."

They extracted nine classes, identified the following relationships: six associations, two composites and one generalization, whereas we were able to extract ten classes, eight associations,two compositions, two generalizations and one dependency. The diagram that our system extracted is included in figure 9.1.

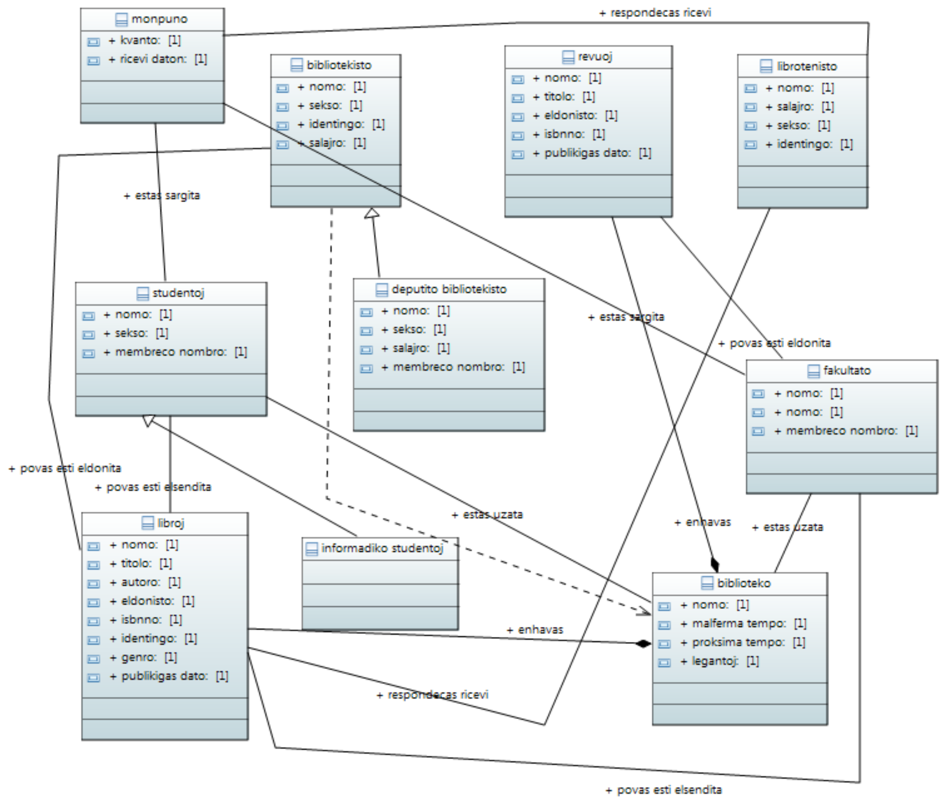
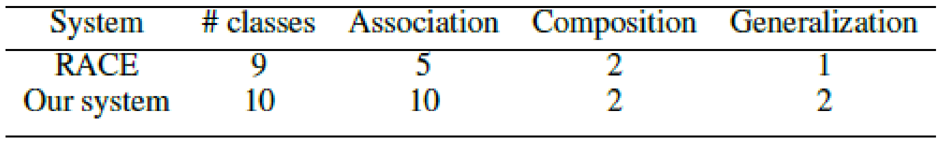


Figure 9.1: Class diagram extracted by our system

Interestingly, the RACE's diagram generated includes several attributes for the classes, that are not explicitly present in the document, and they authors not very clear as to how they identified them. We believe, it may result prejudicial due to the addition of unnecessary information. To compare between the different systems we have taken into the account the number of valid classes, associations, compositions and generalizations identified. We do not know how RACE obtained the attributes they included in their diagram, so we believe comparison between the two systems regarding the attribute validation is unrealistic.



System	# classes	Association	Composition	Generalization
RACE	9	5	2	1
Our system	10	10	2	2

Analyzing the results exposed in Table 1 we have observed that our system extracted an extra class "informadiko studentoj" (informatics students) and generalization relationship between that class and the class "studentoj" (students) that was ignored by RACE system. We also extracted the following extra five association relationships:

- Between "monpuno" (fine) and "fakultato" (faculty)
- Between "monpuno" and "librotenisto" (accountant)
- Between "libroj" (books) and "librotenisto"
- Between "biblioteko" (library) and "fakultato"
- Between "biblioteko" and "studentoj"

In the case of the associations identified only by our system, all are valid except the one between "monpuno" and "fakultato". The reason is that in the requirements is expressed that the fine should only to be charged to students. In summary, we identified an extra class, five more association relationships, which only one of them is amiss, one more generalization relationship, the same composition relationships and a dependency association when compared to the RACE system. Overall, we believe that with this test we proved that it is possible to succesfully apply natural language processing techniques with artificial languages that are not extended. We eased the implementation when auxiliary systems are not available. It will impact on the manteinance and scalability of systems based in languages similar to Esperanto.

9.2 System vs experts

We gave the requirements document used by RACE, in English, to different experts and asked them to create a UML class diagram to the best of their knowledge. One of them is shown in figure 9.2.

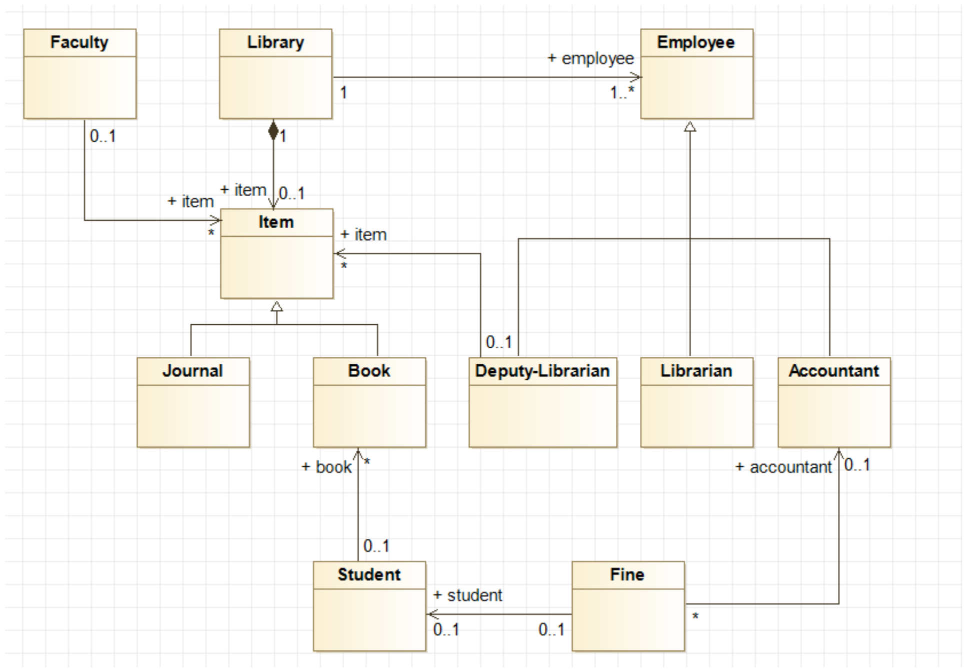


Figure 9.2: Class diagram extracted by expert 1 (E1)

This diagram is a more generic solution, looking at future additions in the system, like the two generic classes, "Item" and "Employee". Nonetheless, all the relevant classes and their relationships have been identified. The rest of the diagrams made by the experts are shown in figure 9.3, figure 9.4, figure 9.5 and figure 9.6 :

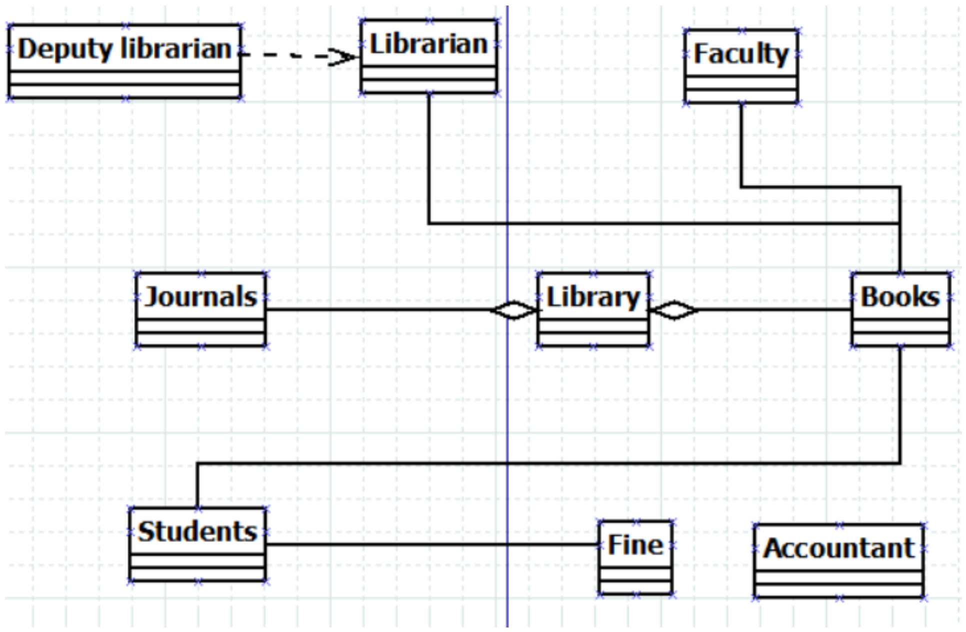


Figure 9.3: Class diagram extracted by expert 2 (E2)

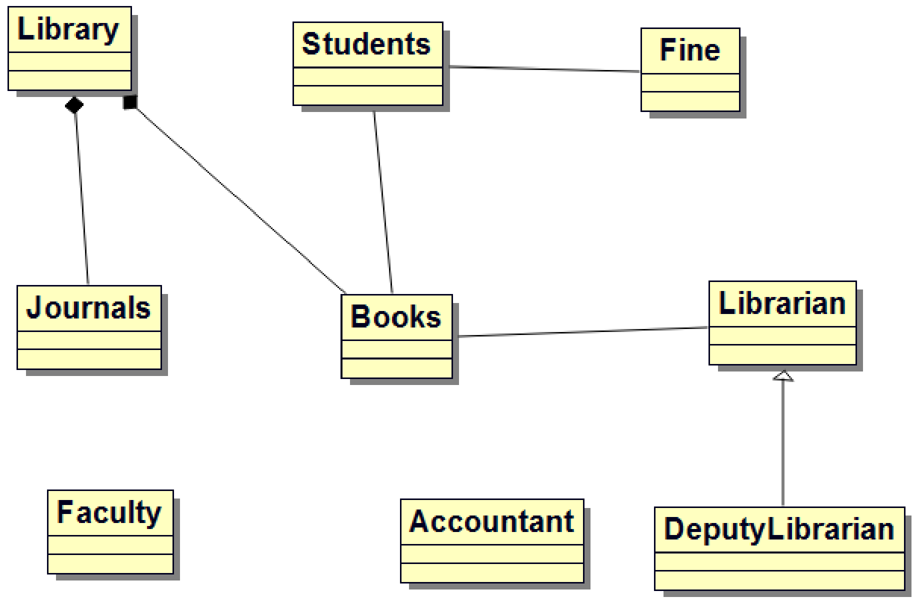


Figure 9.4: Class diagram extracted by expert 3 (E3)

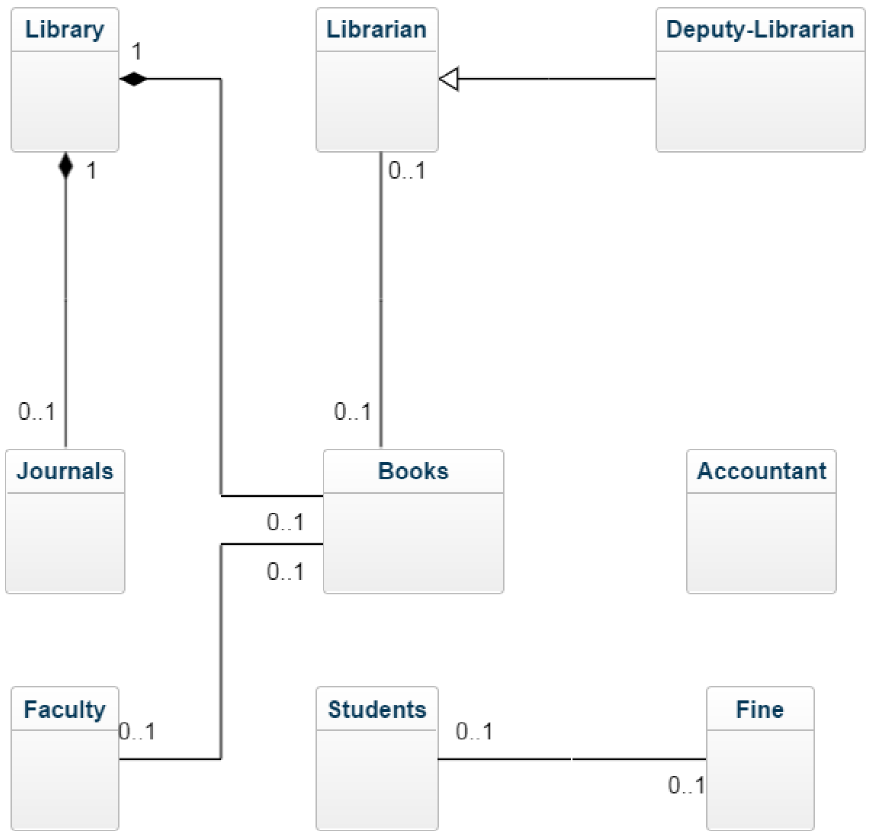


Figure 9.5: Class diagram extracted by expert 4 (E4)

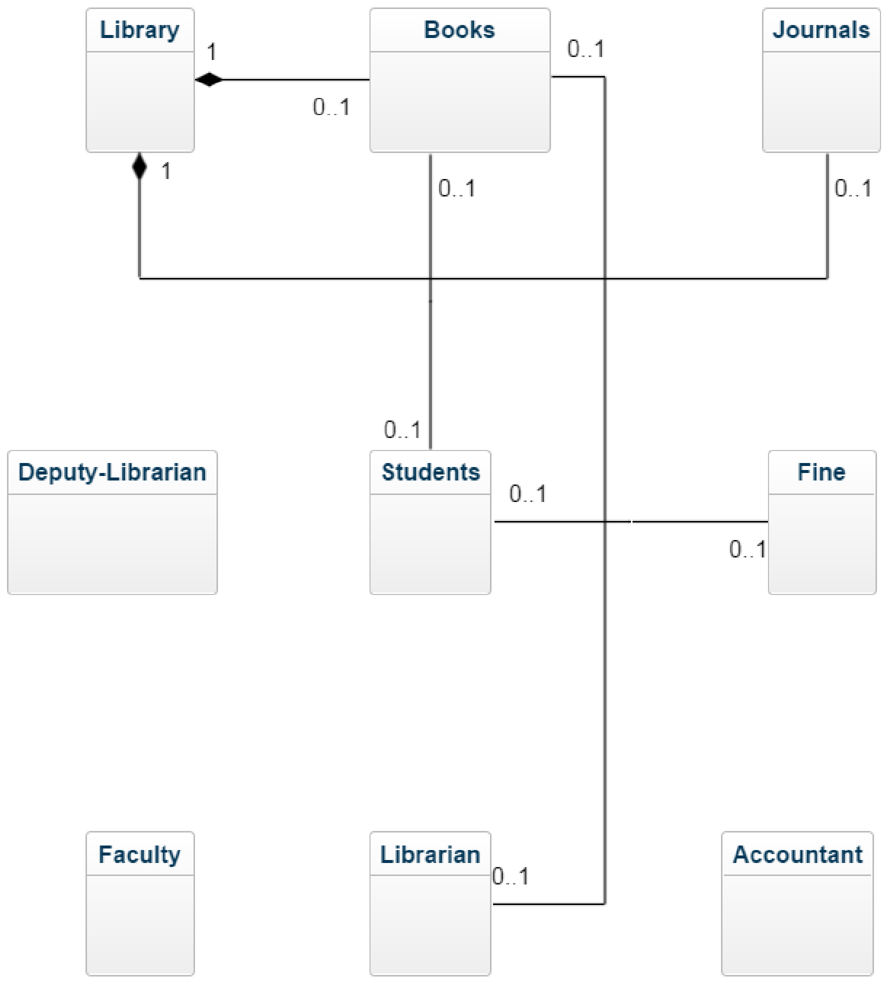
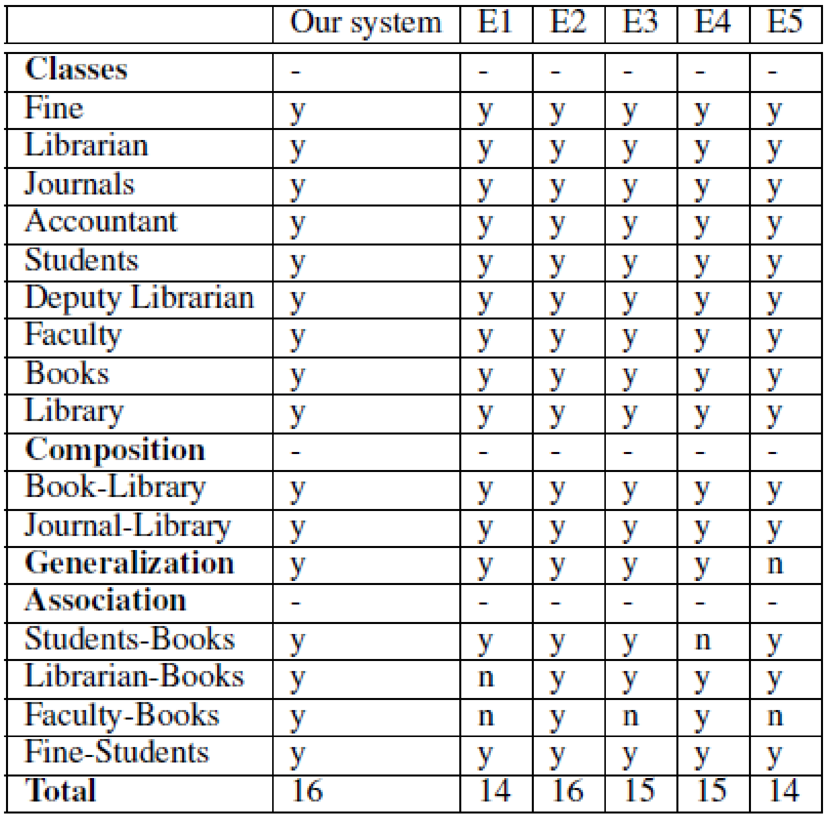


Figure 9.6: Class diagram extracted by expert 5 (E5)

To compare between the expert's diagram and our system's diagram, the following parameters will be analyzed:

- Whether all the relevant classes are identified or not. These classes being: "Fine", "Librarian", "Journals", "Accountant", "Students", "Deputy Librarian", "Faculty", "Books", "Library".
- Check if the composition relationships "Book-Library" and "Journal-Library" exist.

- Check if a generalization relationship between "Librarian" and "Deputy Librarian" exist.
- Check if the following association relationships are present: "Students-Books","Librarian-Books","Faculty-Books" and "Fine-Students".



	Our system	E1	E2	E3	E4	E5
Classes	-	-	-	-	-	-
Fine	y	y	y	y	y	y
Librarian	y	y	y	y	y	y
Journals	y	y	y	y	y	y
Accountant	y	y	y	y	y	y
Students	y	y	y	y	y	y
Deputy Librarian	y	y	y	y	y	y
Faculty	y	y	y	y	y	y
Books	y	y	y	y	y	y
Library	y	y	y	y	y	y
Composition	-	-	-	-	-	-
Book-Library	y	y	y	y	y	y
Journal-Library	y	y	y	y	y	y
Generalization	y	y	y	y	y	n
Association	-	-	-	-	-	-
Students-Books	y	y	y	y	n	y
Librarian-Books	y	n	y	y	y	y
Faculty-Books	y	n	y	n	y	n
Fine-Students	y	y	y	y	y	y
Total	16	14	16	15	15	14

The results have been collected in Table 2, showing that all the relevant classes that our system has identified have also been extracted by the experts, the composition relationships have been all extracted by our system and the experts, the generalization have been identified by the majority of both parties. Nevertheless, there are different results in the identification of the associations by the experts.

9.3 Other evaluation

To provide a more complete evaluation, we have tested our system against a class diagram generated by an expert with wide domain knowledge Fakhroutdinov [2013]. The requirements in English are the following:
"Each customer has unique id and is linked to exactly one account. Account owns shopping cart and orders. Customer could register as a web user to be able to buy items online. Customer is not required to be a web user because purchases could also be made by phone or by ordering from catalogues. Web user has login name which also serves as unique id. Web user could be in several states - new, active, temporary blocked, or banned, and be linked to a shopping cart. Shopping cart belongs to account.Account owns customer orders. Customer may have no orders. Customer orders are sorted and unique. Each order could refer to several payments, possibly none. Every payment has unique id and is related to exactly one account. Each order has current order status. Both order and shopping cart have line items linked to a specific product. Each line item is related to exactly one product. A product could be associated to many line items or no item at all. "

The Esperanto requirements used to generate a diagram in our system are:
"Unu kliento havas : unika identigilo. Kliento porti konto. Konto posedas komercacaro kaj ordonojn. Kliento povis registri kiel retouzanto. Unu retouzanto havas : la ensaluta nomo, nova stato, la blokita stato kaj la malpermesita stato. retouzanto estas ligita al komercacaro. Konto posedas kliento ordonojn. Kliento povas ne ordonojn. Ciu ordonojn povus rilati al pluraj pagoj, eble neniu. Pagoj havas : unika identigilo. Pagoj estas rilatajn al unu konto. Unu ordonoj havas : la ordon statuso. Ordonoj kaj komercacaro esti linioerojn ligita al produkto. La linioelemento esti ligita la produkto. Produkto povus asocii al multaj linioerojn au neniu elemento."

The diagram extracted (in English) for the online shopping case can be shown in figure 9.7. In this case, we can see that many of the attributes and relationships are included in the diagram using domain knowledge.

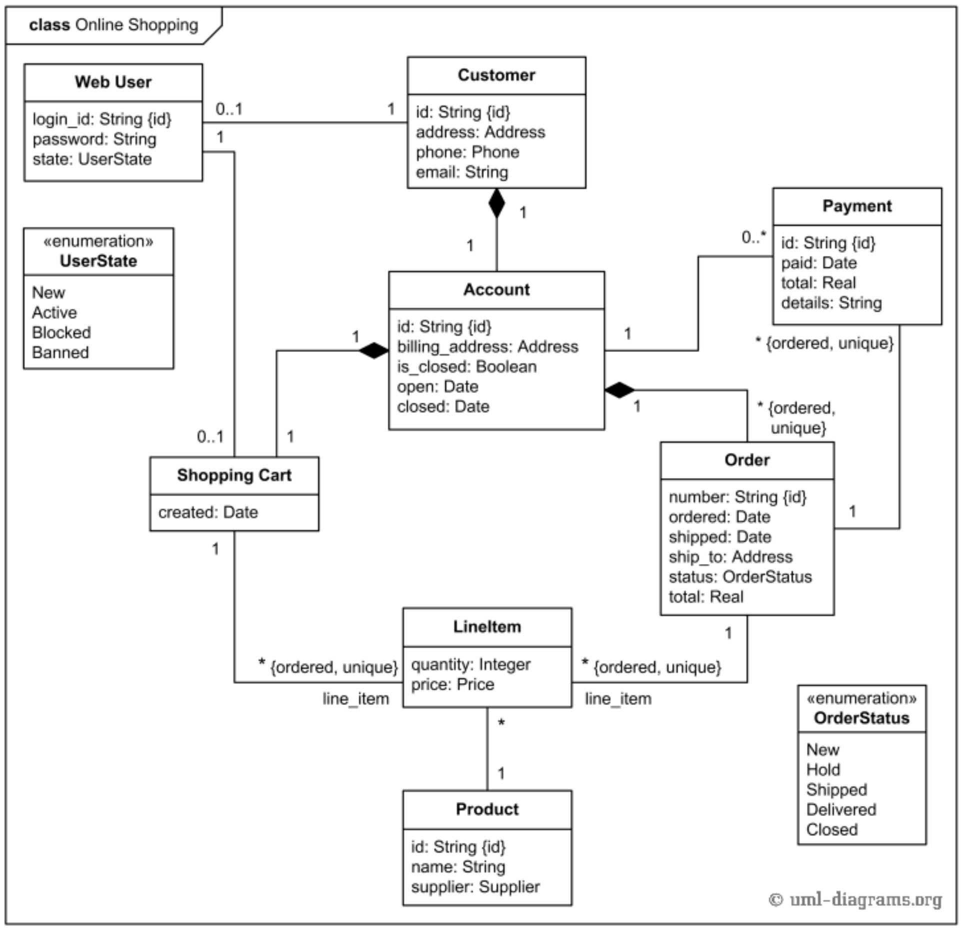


Figure 9.7: Class diagram extracted by an expert for the online shopping case. Created by Fakhroutdinov [2013]

Because of the huge impact that the expert's domain knowledge has in the diagram extracted, we consider the following elements for this evaluation:

- Whether all the relevant classes are identified or not. These classes being: web user, customer, account, payment, order, shopping cart, lineitem, and product.
- Check if the composition relationships "Customer-Account" , "Account-Shopping cart" and "Account-Order" exist.
- Check if the following association relationships are present: "Customer-Web user", "Payment-Account", "Payment-Order", "LineItem-Order",

"LineItem-Product", "Shopping cart-LineItem".

The diagram extracted by our system is shown in figure 9.8 .

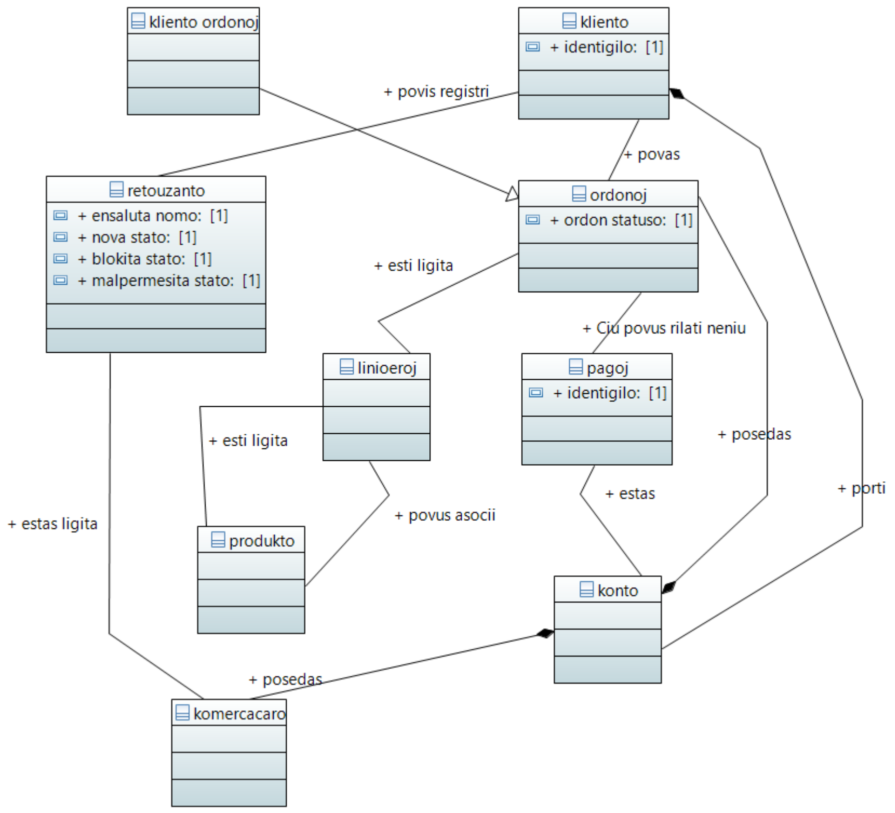


Figure 9.8: Class diagram extracted by our system

We were able to identify nine classes, three composition relationships, one generalization and eight association relationships.

	Expert	Our system
Classes	-	-
Web user	y	y
Customer	y	y
Account	y	y
Payment	y	y
Order	y	y
Shopping cart	y	y
Line item	y	y
Product	y	y
Composition	-	-
Customer-Account	y	y
Account-Shopping cart	y	y
Account-Order	y	y
Association	-	-
Customer-Web user	y	y
Payment-Account	y	y
Payment-Order	y	y
LineItem-Order	y	y
LineItem-Product	y	y
Shopping cart-Line item	y	n
Total	17/17	16/17

Table 9.1: Class diagram extraction comparison between expert with wide domain knowledge and our system

The results have been collected in table 9.1, showing that all the relevant classes that our system has identified have also been extracted by the expert, the composition relationships have been all extracted by our system and the experts, we have identified a generalization between “ordonoj” (Order) and “kliento ordonoj” (Client Order). The association relationships have all been identified by our system, except for the one between “Shopping cart-Line Item”. Despite this, we have identified association relationships that were not included by the expert, some of them are between “Ordonoj-Kliento” (Order-Client), “Retouzanto-komercacaro” (Web user-Shopping cart) and “linieroj-konto” (LineItem-Account).

We conclude this section by stating that our system has successfully generated a class diagram with all the relevant classes implied and many of the association relationships between them, including also generalization relationships and composition relationships. In addition, dependency relationships have also been identified, and our system has been validated by comparison with two expert's diagram and a diagram created by RACE.

10. Conclusions and future work

UML class diagram generation using a requirements document expressed in natural language is challenging. It requires a fine grain approach in design and implementation. We have employed a domain specific language to model the behaviour of an artificial language, Esperanto, in order to facilitate the natural language processing techniques that need to be applied to extract such diagram. We have used the most updated heuristic rules to identify concepts, classes, attributes and relationships. We have developed our prototype using Xtext, as we considered it is the most beneficial tool available at the moment which gives an exhaustive and detailed feedback to the user. We have performed two different evaluations: one comparing our system against another successful system and another comparing the class diagram extracted by our system against those made by experts in the field. Our system provides a friendly environment to the user. Due to our approach, our system is easily extendable and maintainable, making future upgrades to our system straightforward and effortless.

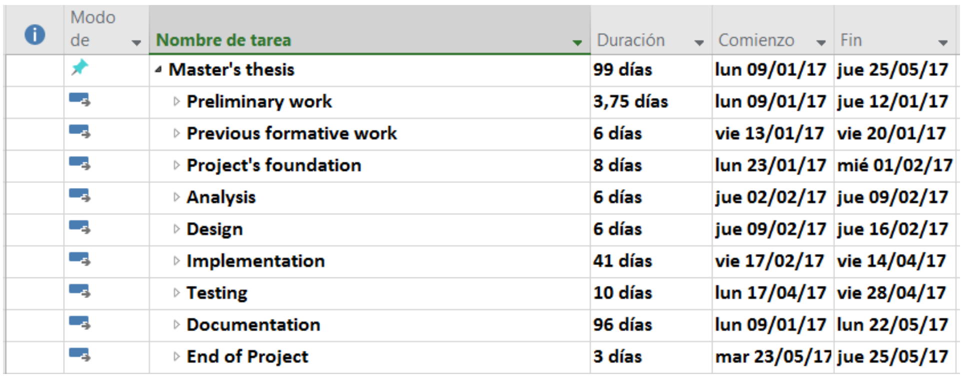
Our system could be improved in a few ways: for example by considering synonyms, so we don't add those elements that already exist in the diagram. The use of hyperonyms to better handle generalizations relationships and the heuristic rules used for relationships identification could be expanded to identify different relationships. The extraction of the methods of classes could also be included. Finally, other types of diagrams could be also developed using our system, such as use case or activity diagrams.

11. Project Planning and Budget

In this section the initial planning and its budget will be included, as well as a later second planning and budget. This allows us to reflect on the differences between the estimated and actual work carried out.

11.1 Initial Planning

This planning was made before starting the project. It includes the different tasks and the estimated time for completion. Also, the initial budget for the planning has been presented.



	Modo de	Nombre de tarea	Duración	Comienzo	Fin
		Master's thesis	99 días	lun 09/01/17	jue 25/05/17
		Preliminary work	3,75 días	lun 09/01/17	jue 12/01/17
		Previous formative work	6 días	vie 13/01/17	vie 20/01/17
		Project's foundation	8 días	lun 23/01/17	mié 01/02/17
		Analysis	6 días	jue 02/02/17	jue 09/02/17
		Design	6 días	jue 09/02/17	jue 16/02/17
		Implementation	41 días	vie 17/02/17	vie 14/04/17
		Testing	10 días	lun 17/04/17	vie 28/04/17
		Documentation	96 días	lun 09/01/17	lun 22/05/17
		End of Project	3 días	mar 23/05/17	jue 25/05/17

Figure 11.1: Overview of the main tasks

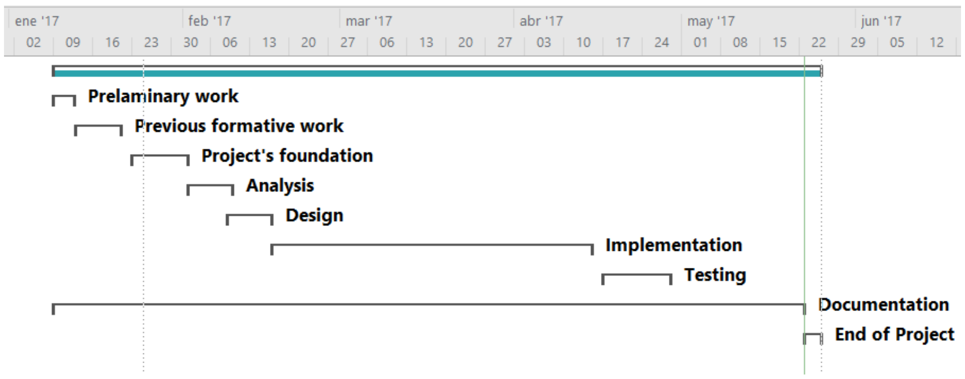


Figure 11.2: Gantt chart of the project's planning

Table 11.1: Description of initial planning tasks.

#	Task	Lv.	Time	Start	End	Description
1	Master's thesis	1	99 d.	09/01	25/05	Master's thesis
1.1	Preliminary work	2	3,75 d.	09/01	12/01	Previous work to be made before starting the project
1.1.1	Project feasibility study	3	6 h.	09/01	09/01	Determine if the project is viable
1.1.2	Alternative analysis	3	6 h.	10/01	10/01	Find out the best alternative
1.1.3	Meeting with the tutor	3	1 h.	11/01	11/01	Meeting with the tutor to establish the definitive subject for the project
1.1.4	Initial drafting of the description of the project	3	6 h.	12/01	12/01	Write the description of the project
1.2	Previous formative work	2	6 d.	13/01	20/01	Work to acquire the knowledge needed to successfully complete the project

1.2.1	Study of DSLs	3	3 d.	13/01	17/01	Getting familiar with DSL
1.2.2	Getting familiar with Xtext	3	3 d.	17/01	19/01	Getting familiar with Xtext
1.2.3	Initial practice with Xtext	3	1 d.	20/01	20/01	Make a first practice with Xtext to confirm that is a valid tool to our project
1.2.4	Milestone: team is familiar with project's subject	3	0 d.	20/01	20/01	The student has the proper knowledge to complete the project
1.3	Project's foundation	2	8 d.	23/01	01/02	Studies that will constitute the foundation of the project
1.3.1	Study and analysis of similar projects	3	2 d.	23/01	24/01	Analyze projects similar to our
1.3.2	State of the art	3	5 d-	25/01	31/01	Perform a complete study of the state of the art in the field
1.3.3	Analysis of the technologies to use in the project	3	1 d.	01/02	01/02	Decide with technologies are more suited for our project
1.4	Analysis	2	6 d.	02/02	09/02	Analysis's tasks of the project
1.4.1	Functional requirements	3	3 d.	02/02	06/02	Find out the functional requirements of the project

1.4.2	Non-functional requirements	3	1 d.	07/02	07/02	Determine the non functional requirements of the project
1.4.3	Use case	3	2 d.	08/02	09/02	Make the main use-case diagrams for the project
1.4.4	Milestone: Project's Analysis	3	0 d.	09/02	09/02	At this time, the analysis of the project is done
1.5	Design	2	6 d.	09/02	16/02	Design tasks of the project
1.5.1	Modular division of the project	3	1 d.	09/02	09/02	Identify the different modules that will compose our system
1.5.2	Design the architecture of the system	3	1 d.	10/02	10/02	Design the system's architecture
1.5.3	Create interactions diagrams for the modules	3	2 d.	13/02	14/02	Model the interactions between the identified modules
1.5.4	Design and create the database	3	1 d.	15/02	05/02	Design a database if deemed necessary
1.5.5	Define the domain model	3	1 d.	16/02	16/02	Create the domain model if necessary
1.5.6	Milestone: Project's Design	3	0 d.	16/02	16/02	At this point, the design of the project is finished
1.6	Implementation	2	41 d.	17/02	14/04	Implementation tasks of the project
1.6.1	Define the subset of Esperanto to use in the project	3	3 d.	17/02	21/02	Find out the subset of Esperanto to use in the project

1.6.2	Create the DSL	3	15 d.	21/02	13/03	Define and implement the DSL to use in the project
1.6.3	Unit tests of the DSL	3	2 d.	14/03	15/03	Unit tests on the DSL
1.6.4	Generate code through the DSL	3	15 d.	16/03	05/04	Code generation through the DSL
1.6.5	Generation code tests	3	2 d.	06/04	07/04	Tests on code generation
1.6.6	Develop the user interface	3	5 d.	10/04	14/04	Develop the user interface to exploit the project's functionality
1.6.7	Milestone: implementation finished	3	0 d.	14/04	14/04	At this point, the implementation of the project is completed
1.7	Testing	2	10 d.	17/04	28/04	Series of tests of different scope
1.7.1	Integration tests	3	4 d.	17/04	20/04	Integration tests on the different systems
1.7.2	Compatibility tests of the modules	3	1 d.	21/04	21/04	Compatibility tests between the different modules
1.7.3	Usability tests	3	3 d.	24/04	26/04	Testing on the user interface
1.7.4	Performance tests	3	2 d.	27/04	28/04	Performance testing
1.7.5	Milestone: Testing completed	3	0 d.	28/04	28/04	At this point, the testing of the project is done
1.8	Documentation	2	96 d.	09/17	22/05	Documentation's tasks of the project
1.8.1	Project's report	3	90 d.	09/01	12/05	Elaborating the report of the project
1.8.2	Technical documentation	3	90 d.	09/01	12/05	Writing the technical documentation of the project

1.8.3	Technical documentation review	3	6 d.	15/05	22/05	Review the documentation
1.8.4	Milestone: Documentation process finished	3	0 d.	24/04	05/05	At this point, the documentation of the project is finished
1.8.5	Article for JRC	3	10 d.	08/05	09/05	Writing the article for a journal
1.8.6	Review of the article	3	2 d.	10/05	10/05	Review the article
1.8.7	Milestone: Article's submission	3	0 d.	10/05	10/05	At this point, the article should be submitted
1.9	End of Project	2	3 d.	23/05	25/05	End of project tasks
1.9.1	Project submission with all its documentation	3	1 d.	23/05	23/05	Deliver the project and its documentation
1.9.2	Deployment	3	1 d.	24/05	24/05	Deployment the project
1.9.3	Backup of the code and the documentation	3	1 d.	25/05	25/05	Make a backup of the code and documentation
1.9.4	Milestone: End of project	3	0 d.	25/05	25/05	End of project

To elaborate the budged the following factors have been taken into account: the salary that a professional team would earn, different costs, like the ones incurred by hardware needs and monthly costs, like maintenance and renting space for the offices.

The table 11.2 contains an estimation of the salary according to each workers' profile.

Profile	€/hour
Analyst	50
Designer	50
Coder	30
Tester	40
System's administrator	35
Project manager	70

Table 11.2: Salaries by profile.

The table 11.3 shows the estimated allotted time for each worker according to their profile on the project.

Profile	Allotted time (hours)	Cost (€)
Analyst	308	15400
Designer	96	4800
Coder	248	7440
Tester	112	4480
System's administrator	40	1400
Project manager	21	1700
Total	825	**34990 €**

Table 11.3: Time and cost by each worker.

The shows the different costs dedicated to hardware.

Hardware	Price (€)
Asus computer	950
Web server	400
Total	**1350 €**

Table 11.4: Hardware costs.

In are reflected the fixed monthly expenses, including costs of Internet access, electricity and heating.

Cost	Duration (months)	Price/month	Total
Internet access	5	40	200
Electricity	5	150	750
Heating	5	100	1500
Total			**1500 €**

Table 11.5: Fixed monthly expenses

Taking into consideration the different expenses presented, the total budged of the project is **37.840€**.

11.2 Final Planning

The initial planning has suffered some changes as the work has been done. We will explain the changes that have taken place and the reason why.

- 1.5.1 modular division of the project: as we learned the technologies to employ, we realized that a modular division will not have much sense in our project.
- 1.5.4 design and create the database: initially, we were not sure if we would need a database in our project, as we developed our system we realized that a database would not have any positive impact in our system.
- 1.6.3 unit tests of the dsl and 1.7 testing: after discussing with the tutor of the project, it was decided that unit tests and other types of tests on the DSL would imply a lot of work and a very few valuable feedback, so we decided not to include them. Instead, we dedicated our efforts to test with different inputs analyzing the diagrams generated.

The repercussions of this changes in the budged are shown in table 11.6 :

Profile	Allotted time (hours)	Cost (€)
Analyst	300	15000
Designer	80	4000
Coder	248	7440
Tester	16	640
System's administrator	32	1120
Project manager	21	1700
Total	825	**29000 €**

Table 11.6: Time and cost by each worker.

The rest of the budged did not change, so the final budged ascends to **32750 €**.

12. Dissemination of results

This is the main article of this master's thesis. It contains all the findings and most of the work carried out in the project. In this article we present our system and we remark the possible applications to extract UML diagrams by processing an artificial language, Esperanto, using natural language processing techniques. We provide support to the users through a customized text editor.

In order to choose the best suited journal to submit our article, we performed an analysis of the existing journals related to our field.

Initially we considered two journals inscribed into the Journal Citation Report (JCR).

International Journal on Software and Systems Modeling (figure 12.1)

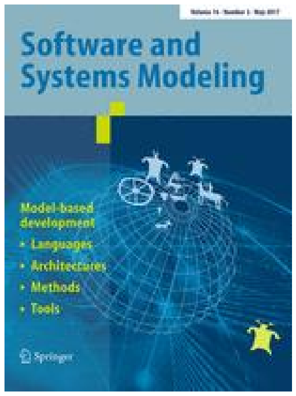


Figure 12.1: International Journal on Software and Systems Modeling (SoSyM)

- Software and Systems Modeling Website
- Editorial: Springer
- Impact factor: 0.990
- Impact factor (5 years): 1.497
- Main interests: theoretical and practical issues in the development and application of software and system modeling languages, techniques, and methods, such as the Unified Modeling Language.

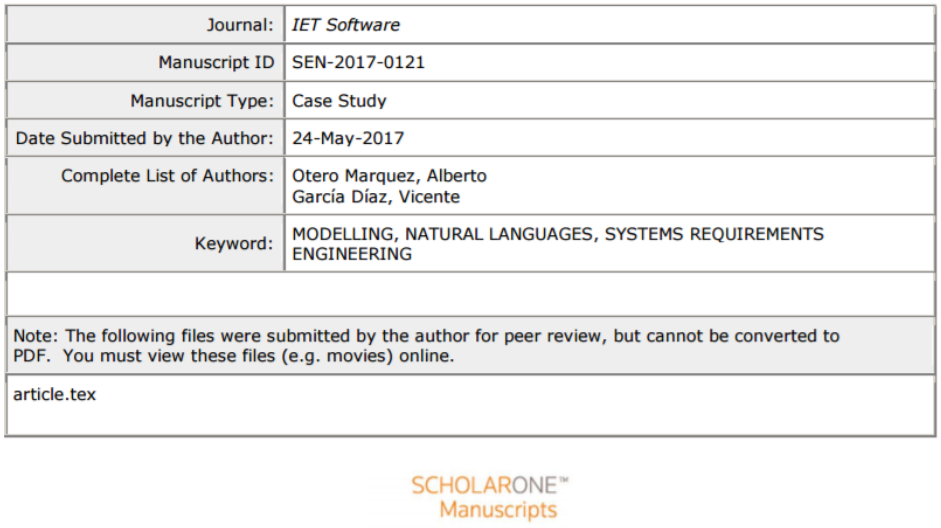
IET Software (figure 12.2)

Figure 12.2: International Journal on Software and Systems Modeling (SoSyM)

- IET Software Website
- Editorial: Institution of Engineering and Technology (IET)
- Impact factor: 0.473
- Impact factor (5 years): 0.519
- Main interests: all aspects of the software lifecycle, including design, development, implementation and maintenance, specially the methods used to develop and maintain software, and their practical application.

Our preferred option was initially the "Software and Systems Modeling" journal, given their higher impact factors, but after reviewing our article, we realized that our article was better suited for the "IET Software" journal, as our solution touched many different fields, not only the modeling area, so we went with IET Software, as it included more generic IT solutions.

Our paper was submitted on 24/05/17 to IET Software. its status is "Under Review". In figure 12.3 and figure 12.4 can be seen the status of the submission.

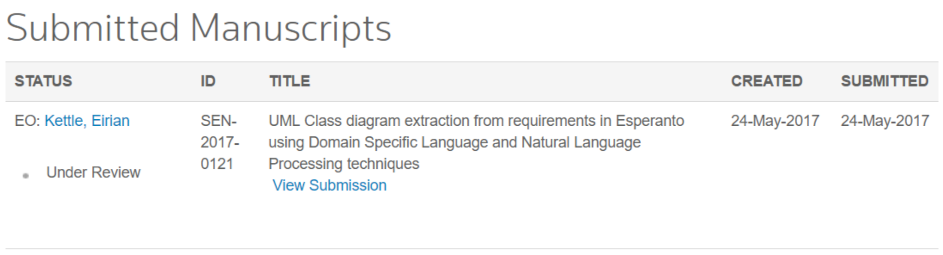


Journal:	*IET Software*
Manuscript ID	SEN-2017-0121
Manuscript Type:	Case Study
Date Submitted by the Author:	24-May-2017
Complete List of Authors:	Otero Marquez, Alberto García Díaz, Vicente
Keyword:	MODELLING, NATURAL LANGUAGES, SYSTEMS REQUIREMENTS ENGINEERING

Note: The following files were submitted by the author for peer review, but cannot be converted to PDF. You must view these files (e.g. movies) online.

article.tex

Figure 12.3: Proof of submission I

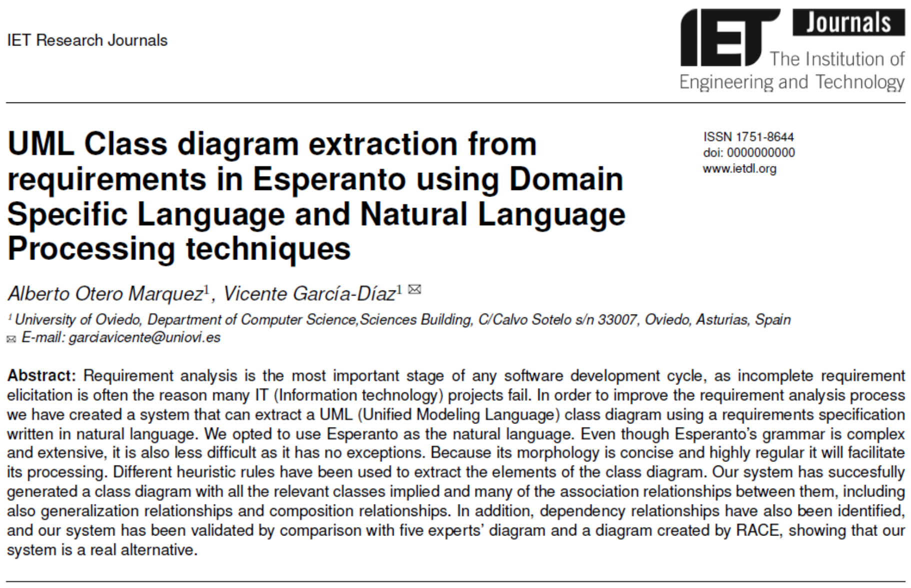


Submitted Manuscripts

STATUS	ID	TITLE	CREATED	SUBMITTED
EO: Kettle, Eirian • Under Review	SEN-2017-0121	UML Class diagram extraction from requirements in Esperanto using Domain Specific Language and Natural Language Processing techniques View Submission	24-May-2017	24-May-2017

Figure 12.4: Proof of submission II

A. Article



IET Research Journals

ISSN 1751-8644
doi: 0000000000
www.ietdl.org

UML Class diagram extraction from requirements in Esperanto using Domain Specific Language and Natural Language Processing techniques

Alberto Otero Marquez[1], *Vicente García-Díaz*[1] ✉
[1] *University of Oviedo, Department of Computer Science,Sciences Building, C/Calvo Sotelo s/n 33007, Oviedo, Asturias, Spain*
✉ *E-mail: garciavicente@uniovi.es*

Abstract: Requirement analysis is the most important stage of any software development cycle, as incomplete requirement elicitation is often the reason many IT (Information technology) projects fail. In order to improve the requirement analysis process we have created a system that can extract a UML (Unified Modeling Language) class diagram using a requirements specification written in natural language. We opted to use Esperanto as the natural language. Even though Esperanto's grammar is complex and extensive, it is also less difficult as it has no exceptions. Because its morphology is concise and highly regular it will facilitate its processing. Different heuristic rules have been used to extract the elements of the class diagram. Our system has succesfully generated a class diagram with all the relevant classes implied and many of the association relationships between them, including also generalization relationships and composition relationships. In addition, dependency relationships have also been identified, and our system has been validated by comparison with five experts' diagram and a diagram created by RACE, showing that our system is a real alternative.

1. Introduction

Natural language is the most frequently used language for expressing requirements, as it is common to customers, users and requirements engineers. Due to the ambiguity of natural language, which is liable for different interpretations influenced by geographical, psychological and sociological factors, a good requirement gathering process in natural language is usually difficult to perform Resnik et al. [1999]. An incomplete requirement elicitation is often the main reason that leads IT projects to failure. Projects with missing requirements will cause dissatisfaction to the customer, and incomplete products will require more time and resources to be finished than if originally planned Bergey et al. [1999]. The importance of performing an exhaustive and complete requirement list on the early stages of a project is critical. Therefore, a support tool to automatize these tasks would be highly valuable and desirable. In recent years, Object-Oriented (O-O) Analysis and Design has become the mainstream trend for software development. A class is an abstraction of representative real-world objects. A class models the behavior of objects that are constrained by the same rules Starr and Stephen

[2001] and share attributes and behaviors. Classes are arranged into a class diagram. A class diagram in the Unified Modeling Language (UML) represents the classes used and their relationships in the system. Those classes act as the vocabulary for O-O system, model simple collaborations and are the foundation for the logical database design Booch et al. [1999]. UML class diagrams are at the core of O-O analysis and identifying the classes that model the requirements seems crucial. In fact, it is considered by some authors as the most important skill in developing a O-O system Grady [1994]. Hence, the automated generation of class diagrams will save time and effort to a system analyst, specially for beginners. Nevertheless, the automation of class generation from a written text in natural language is extremely challenging due to the following reasons Richter [1999] Maciaszek [2001]:

- Natural languages are ambiguous and full of exceptions. Hence, rigurous and precise analysis proves quite difficult.
- Any given semantic can be represented in many different ways.
- Concepts that are not explictly expressed in the written source are often difficult to identify. It requires an expert domain knowledge to find the hidden classes.

Our approach in this paper tries to address the problem presented but it does not try to resolve it completely. We believe that by using Esperanto, an artificially made language with no grammatical exceptions, most part of the ambiguity would be suppresed. A Domain Specific Language (DSL) has been developed to model the most common behavior of the Esperanto language. Natural Language Processing (NLP) techniques will be applied to the requirements document in Esperanto and different heuristic rules will be used to extract a UML Class diagram, which will contain the structure of a suggested system that satisfies the requirements. We have used NLP techniques and heuristic rules to address the following research questions: (1) How to identify concepts, (2) how to discern between candidate classes and attributes for each concept and (3) how to extract relationships between classes, such as generalization, dependency, associations and compositions. The rest of the paper is organized as follows: Section 2 will cover the literature review and related works, Section 3 presents the model developed, Section 4 briefly introduces the different heuristic and external tools used to generate the diagram,Section 5 contains implementation details, Section 6 contains the evaluation and discussion of our system and Section 7 contains the conclusions of this paper and future works.

2. State of the Art

2.1 Natural Language Processing to extract some Unified Modeling Language (UML) diagram

Different approaches have been used to analyze natural language requirements. Ibrahim and Ahmad [2010] Deeptimahanti and Babar [2009] Letsholo et al. [2013] Zhou and Zhou [2004] Song et al. [2004] However, a few have been oriented on class diagram generation. In this section we review the works that use NLP or domain ontologies to analyze natural language requirements, with emphasis in the ones that focus in class diagram extraction. Ibrahim, Mohd et al Ibrahim and Ahmad [2010] developed the Requirements Analysis and Class diagram Extraction (RACE) method, which combines the use of NLP techniques and domain ontologies to extract a class diagram from a given set of informal requirements. The components used in this system are: (1) OpenNLP as a parser, (2) a Stemming Algorithm; responsible of abbreviating words by removing affixes and suffixes, (3) WordNet; in charge of providing semantic validation during the process, (4) domain specific Ontology Library; to enhance the concept's identification process and (5) Class Extraction Engine, which uses heuristic rules to extract the class diagram. The results obtained in their case study are promising. The system was able to find concepts based on nouns, verbs and noun phrases as well as identifying four different types of relationships: Generalization, Association, Composition, and Aggregation. Nevertheless, it failed to identify one to one, one to many, and many to many relationships. Deeptimahanti, Deva Kumar et al Deeptimahanti and Babar [2009] developed UML Model Generator from Analysis of Requirements (UMGAR), a tool capable of generating Use-case diagrams, analysis and design class model and collaboration diagrams by processing requirements expressed in natural language, doing it in an automatized way. The key components of this system are: (1) Stanford Parser; used to extract information from each requirement, avoiding the use of different single components for tasks such as stemming and tagging tools, (2) WordNet; which provides morphological analysis and plural to singular conversion and (3) JavaRAP; which provides pronoun's treatment. Ultimately, UMGAR is a domain independent tool capable of generating different UML models from requirements expressed in natural language, offering also Extensible Markup Language (XML) support for the visualization of the diagrams created. Letsholo, Keletso J et al Letsholo et al. [2013] designed Textual Requirements into Analysis Models (TRAM), a tool able to assist in the automatic creation of analysis models from requirements expressed in natural languages. It offers three advantages over similar existing tools, (1) it closes the gap between unstructured natural language requirements and its analysis models by providing a series of conceptual patterns, named Semantic Object Models (SOMs), (2) it offers requirements traceability to assist analysts and developers to locate errors and (3) it validates and improves the quality of the model generated by the domain expert or software analyst by using a question and answer method. Zhou, Xiaohua et al Zhou and Zhou [2004] proposed to use NLP techniques to handle the written requirements and domain knowledge through ontologies to improve the performance of class identification. Their designed system used two inputs: (1) text functional specifications and (2) structured domain ontologies. Its output is a class diagram, which includes attributes of each class and inter-class relationships. They were able to identify the following relationships; generalization, aggregation and association. Furthermore, they classified the association relationships into one-to-one, one-to-many and many-to-many. The core components of the system are: (1) Transformation Tagger for part-of-speech tagging, (2) Link Grammar as the sentence parser and (3) WordNet for semantic validation. They outlined a possible improvement in the diagrams generated by employing ontologies. Song, Il-Yeol et al Song et al. [2004] proposed a Taxonomic Class Modeling (TCM) methodology. It was able to apply noun analysis, class categorization, sentence structure rules, checklists and heuristic rules for modeling. In their approach, they were able to identify: (1) classes from concepts stated as noun phrases in the requirements statement, (2) classes stated as a verb phrase in the requirements statement and (3) hidden classes that were not explicitly stated in the requirements statement by applying domain knowledge to the class categories. They also employed WordNet as many of the works previously presented. They found that the TCM methodology is effective to identify domain classes for object-oriented applications in different domain fields. They state that their solution is practical as it can be easily and effectively applied to different project domains. We recognized the approach in Ibrahim, Mohd et al Ibrahim and Ahmad [2010] as the most relevant to our work, considering the exhaustive and complete heuristic rules used for concepts, attributes, classes and relationships identification.

2.2 Why Esperanto?

Originally, English was considered for our system, but it implied the use of multiple systems to perform parsing, stemming, semantic validation and more, which are not available for Esperanto. By using Esperanto external components are not required, as the parsing and other related tasks will be performed by ourselves. Besides, there are a couple more reasons that make Esperanto the most preferable choice as the language to work in this project:

1. To apply traditional parsing a complete formal grammar of the language is required, being context-free and unambiguous. Natural languages, like English or Spanish, do not meet those requirements. These languages are complex and full of exceptions. Esperanto can be considered as a natural language for our purposes, as it includes all the common word types and grammatical features. Even though Esperanto's grammar is complex and extensive, it is also less complicated due to a complete lack of exceptions. Its morphology is concise and highly regular, which will facilitate its parsing Aasgaard [2006].

2. The basics of the Esperanto grammar are covered by 16 rules, which simplify and speed up the learning and processing of the language Forster [1982].

There are also some disadvantages by using Esperanto, namely:

1. Esperanto is not widely extended, being spoken or understood by a few millions of people.

2. Only users with knowledge of Esperanto would be able to use our system.

In addition, some parsers have been already developed for Esperanto. They take advantage of the highly regular morphology of Esperanto, its design to avoid ambiguity and the total lack of exceptions. These parsers are EspGam Bick [2007] and EOParser GermaneSoftware [2006]. EspGam is a Constraint Grammar parser for Esperanto. It is a rule based system that applies contextual rules to tackle morphological disambiguation and perform syntactic analysis. The system contains a lexicon of 28.000 lexemes built from data of a bilingual Esperanto-Danish dictionary and a Danish-Esperanto machine translation system. The disambiguation and syntactic rules are formulated by removing, selecting or mapping category tags, based on sentence-wide context conditions. They measured the performance of their parser against a hand-annotated standard corpus of news text produced in Esperanto. The parser accuracy rates are of 99.5% for part-of-speech (POS) and 92,1% for syntactic dependency. EOParser is a morphology parser written in Ruby for Esperanto. EOParser offers a text-based User Interface for querying, and it can also be used as a library. The parser accepts Esperanto sentences and returns the translation as a string based output and the morphological properties as long as it understands the semantic meaning of the query.

3. The Model

We developed a Domain Specific Language (DSL) to implement the most common structures of the language Esperanto. We decided to use Xtext Foundation [2006] as it is one of the most common tool employed to create textual DSLs. That way we could benefit from its features, such as the generation of a parser, serializer and a smart editor. The DSL that we developed is actually a textual editor for the Esperanto language, to be more precise, to a big subset of the Esperanto language. Another reason to use Xtext is the fact that Xtext uses and integrates with the Eclipse Modeling Framework Steinberg et al. [2008], a framework and code generation tool for building Java applications based on model definitions. We provided further customization for the smart editor, so the user would feel more comfortable when typing his requirements. In that way, we have unified three technologies, Java, XML and UML.

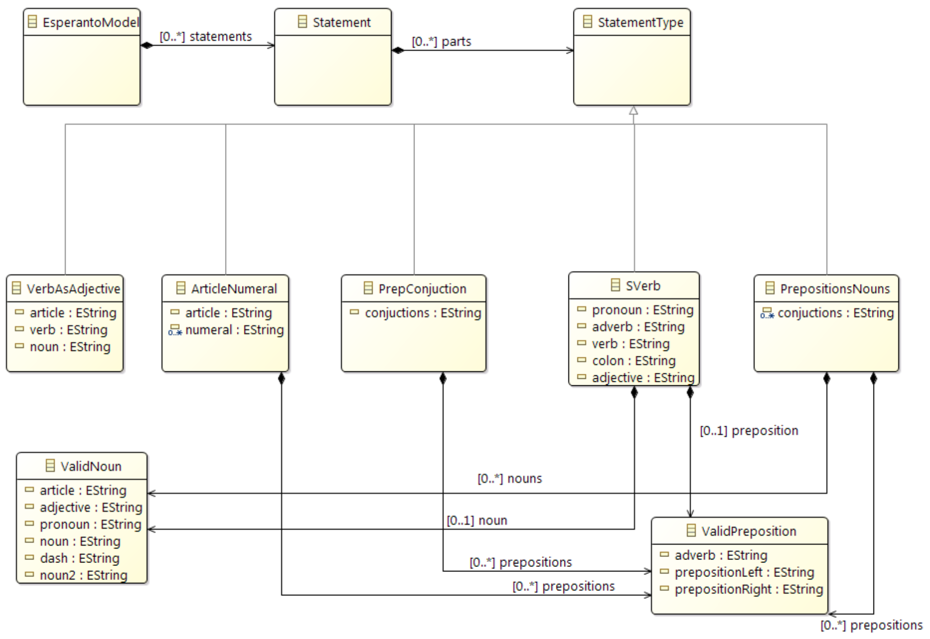


Figure A.1: Metamodel that implements the Esperanto grammar.

The model created has been represented in Figure A.1. EsperantoModel contains a series of Statements, where each Statement is composed of different parts, which are of type StatementType. A StatementType can be of type PrepositionsNouns, Sverb, PrepConjuction, VerbAsAdjective, ArticleNumeral, SpecialCharacter or Questions and so on. This model contains the behaviour of the most common type of structures in the Esperanto language. Figures A.2 and A.3 illustrate the type of customization that can be achieved into the graphical editor, where each type of structure has its own color. In the outline, every member of each structure is categorized according to their type.

```
La biblioteko sistemo estas uzata de la informadiko studentoj kaj fakultato.
La biblioteko enhavas librojn kaj revuoj.
Libroj povas esti elsendita al kaj la studentoj kaj fakultato.
Revuoj nur povas esti eldonita al la fakultato.
Libroj kaj revuoj nur povas esti eldonita fare de la bibliotekisto.
La deputito bibliotekisto estas komandata de ricevi la reiris libroj kaj revuoj.
La librotenisto respondecas pri ricevi la monpuno por posttempa libroj.
La monpuno estas sargita nur al studentoj, kaj ne al la fakultato.
```

Code. A.2: Sample of the customized editor view with some Esperanto text

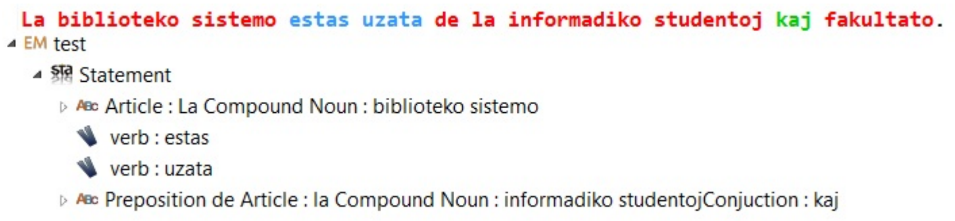

La biblioteko sistemo estas uzata de la informadiko studentoj kaj fakultato.
EM test
Statement
Article : La Compound Noun : biblioteko sistemo
verb : estas
verb : uzata
Preposition de Article : la Compound Noun : informadiko studentojConjuction : kaj

Fig. A.3: Outline view customization

4. Extraction of UML Class Diagram

The first step needed to obtain a class diagram from natural language requirements is to identify the concepts present in it. To identify and extract concepts we will:

1. Identify the stop words. In our case, we found that typically these are usually prepositions, articles, pronouns and conjuctions.

2. Calculate the total number of words T in the document without the stop words, the number of occurrences O_w of each word, and then

calculate the frequency F of each word, according to:

$$F = O_{\mathrm{w}}/T \tag{A.1}$$

3. Save the identified proper nouns, noun phrases and verbs as concepts.

4.1 Class Identification

After extracting the concepts, we can process them to extract candidate classes by applying the following rules:

1. If a concept only appears once in the document and its frequency is less than 2%, then it will be ignored.
2. If a concept is related to the design elements, then it will be ignored.
3. If a concept is related to Location name or People name, then it will be ignored.
4. If a concept is an attribute, then it will be ignored.
5. If a concept is a noun phrase (noun + noun) and the secound noun is an attribute, then the first noun is a class and the second one is an attribute of that class.
6. If a concept does not satisfy any of the rules, then it's likely a class.

4.2 Attribute Identification

The following rules assisted us in extracting attributes:

1. If a class is followed by the verb "have/has" and a colon, then the following enumeration is a list of attributes of the class.
2. If a concept is a noun phrase (noun + noun) and the secound noun is an attribute, then the first noun is a class and the second one is an attribute of that class.

4.3 Relationship Identification

Performing verb analysis and applying the next rules we have extracted relationships:

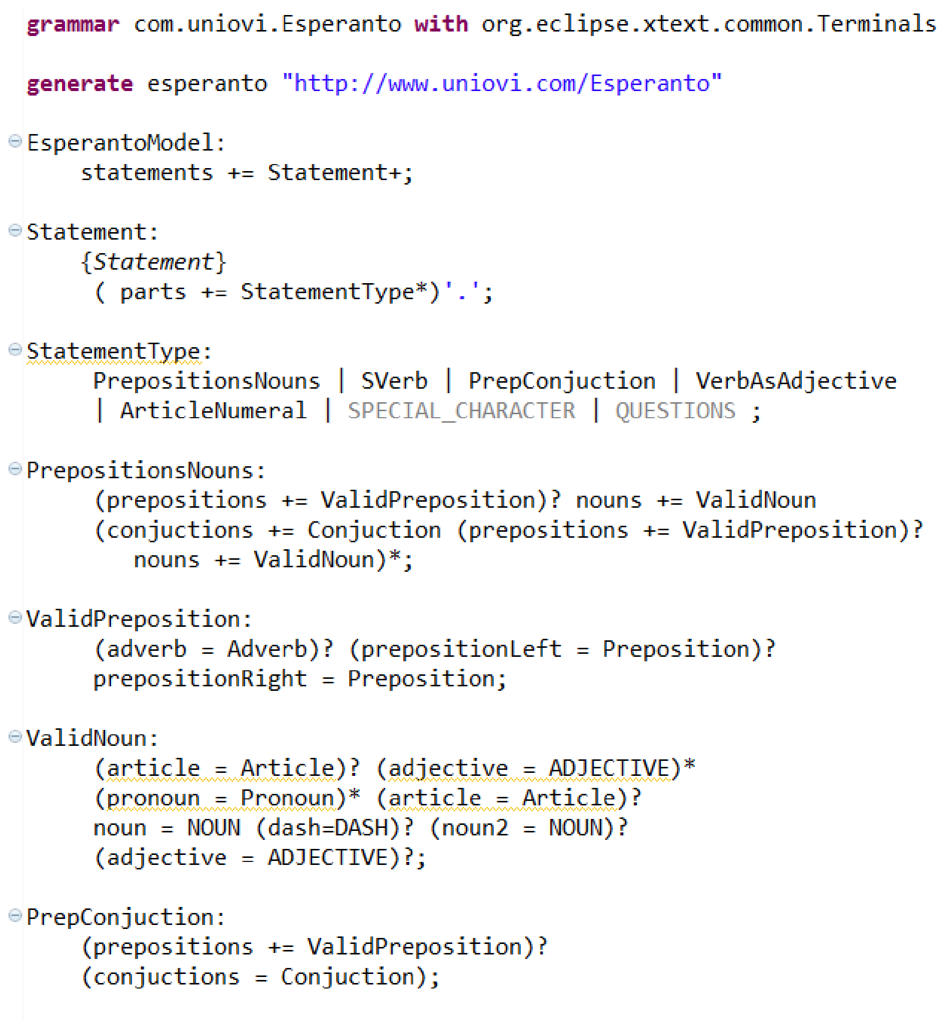
1. If we have a compound name (noun + noun) and the second one is a class, then the compound name is a generalization to the already identified class (second noun).
2. If the concept is a verb, and we can find a sentence in the from C1-V-C2, where C1 and C2 are classes, then V is an Association relationship.
3. If the concept is a verb, satisfies Relationship identification Rule 2 and the verb is one of the following (in Esperanto) ["require", "depends on", "rely on", "based on", "uses", "follows"] then the relationship discovered is a Dependency relationship.
4. Given a sentence in the form C1 + R1 + C2 + AND + C3 where C1, C2 and C3 are classes and R1 is a relationship, then the relationship R1 applies between the classes C1 and C2 and between C1 and C3.

5. Implementation

In this section, we will show the role of the technologies employed in our system. As previously stated, Xtext has been the tool chosen to develop our system. Xtext is used to create the DSL that contains the grammar of the Esperanto language. When creating a new Xtext project, the following projects are created:

- A project that contains the grammar definition and all language-specific components (parser,lexer,linker,validation,etc.)
- A project to run unit tests for the language
- A project to manage platform-independent IDE functionality (e.g. services for content assist)
- A project that contains the eclipse editor and other workbench related functionality
- A project to perform unit test for the editor.

We use Xtext's grammar to define our Esperanto DSL (see Figure A.4). We also have a "mwe2" extension file where we can customize several options about our project, such as if we want to generate a serializer, a validator, the extension files for our language, and so on. In our case, we used two generators, one to generate a text file with the processed output of a user's requirements and another to generate the UML Class diagram. We have used Java and Xtend for these generators. The behaviour of Xtext and its generators is that once the user saves the changes made in his editor, the generators are automatically executed, generating the different outputs when the user saves.



```
grammar com.uniovi.Esperanto with org.eclipse.xtext.common.Terminals

generate esperanto "http://www.uniovi.com/Esperanto"

EsperantoModel:
    statements += Statement+;

Statement:
    {Statement}
     ( parts += StatementType*)'.';

StatementType:
     PrepositionsNouns | SVerb | PrepConjuction | VerbAsAdjective
     | ArticleNumeral | SPECIAL_CHARACTER | QUESTIONS ;

PrepositionsNouns:
     (prepositions += ValidPreposition)? nouns += ValidNoun
     (conjuctions += Conjuction (prepositions += ValidPreposition)?
        nouns += ValidNoun)*;

ValidPreposition:
     (adverb = Adverb)? (prepositionLeft = Preposition)?
     prepositionRight = Preposition;

ValidNoun:
     (article = Article)? (adjective = ADJECTIVE)*
     (pronoun = Pronoun)* (article = Article)?
     noun = NOUN (dash=DASH)? (noun2 = NOUN)?
     (adjective = ADJECTIVE)?;

PrepConjuction:
    (prepositions += ValidPreposition)?
    (conjuctions = Conjuction);
```

Fig. A.4: Fragment of the grammar definition in Xtext

We also customized the behavior of the editor's syntax highlighting and the outline view, for the latter, we displayed the simple elements that compose the grammatical structures. Furthermore, we used the UML2 package to create the diagram and its different components Kenn.Hussey [2011].

6. Evaluation and discussion

We have used two methods to evaluate our system: we have compared our system against the approach proposed by Ibrahim, Mohd et al Ibrahim and Ahmad [2010] and we have tested the diagram generated by our system against diagrams extracted by fellow engineers.

6.1 System vs system

In order to validate our system and obtain a general assessment for accuracy and efficiency of our system, we have tested it against the RACE tool developed by Ibrahim and Ahmad [2010]. We have used the same input requirements, a library system, with two changes:

1. We have added the following line (in Esperanto) to provide a dependency example: "The librarian depends on the library".

2. We have added attributes to the requirements. As we have not employed ontologies, only the attributes present in the text could be extracted into the diagram.

This is the original input used by RACE (English):
"The library System is used by the Informatics students and Faculty. The Library contains Books and Journals. Books can be issued to both the Students and Faculty. Journals can only be issued to the Faculty. Books and Journals can only be issued by the Librarian. The deputy-Librarian is in-charge of receiving the Returned Books and Journals. The Accountant is responsible for receiving the fine for over-due books. Fine is charged only to students, and not to the Faculty."
Whereas our input used for the testing (Esperanto) is:
"La biblioteko sistemo estas uzata de la informadiko studentoj kaj fakultato.La biblioteko enhavas librojn kaj revuoj.Libroj povas esti elsendita al kaj la studentoj kaj fakultato.Revuoj nur povas esti eldonita al la fakultato.Libroj kaj revuoj nur povas esti eldonita fare de la bibliotekisto.La deputito bibliotekisto estas komandata de ricevi la reiris libroj kaj revuoj.La librotenisto respondecas pri ricevi la monpuno por posttempa libroj.La monpuno estas sargita nur al studentoj, kaj ne al la fakultato. La bibliotekisto dependas de la biblioteko.Unu libroj havas: nomo, titolo, autoro, eldonisto, isbnno, identingo, genro kaj la publikigas dato.La biblioteko havas: nomo, malferma tempo,proksima tempo kaj legantoj.La studentoj havas: nomo, sekso kaj membreco nombro.La deputito bibliotekisto havas: nomo, sekso,

salajro kaj membreco nombro.La fakultato havas: nomo kaj membreco nombro.Fakultato estas savita fakultato nomo, kaj membreco nombro.La monpuno havas: kvanto kaj la ricevi daton.Bibliotekisto estas konservita bibliotekisto nomo, bibliotekisto salajro, bibliotekisto sekso kaj bibliotekisto identingo.Unu revuoj havas: nomo, titolo, eldonisto, isbnno kaj la publikigas dato.Unu librotenisto havas: nomo, salajro, sekso kaj identingo." They extracted nine classes, identified the following relationships: six associations, two composites and one generalization, whereas we were able to extract ten classes, eight associations,two compositions, two generalizations and one dependency. The diagram that our system extracted is included in Figure A.5.

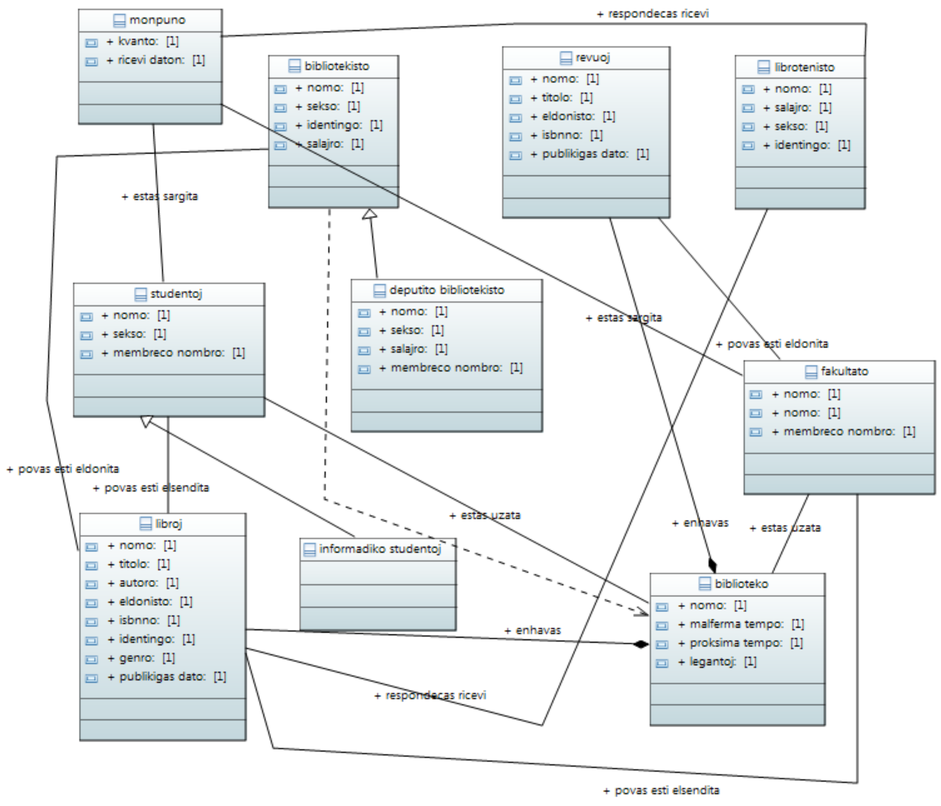


Fig. A.5: Class diagram extracted by our system

Interestingly, the RACE's diagram generated includes several attributes for the classes, that are not explicitly present in the document, and they authors not very clear as to how they identified them. We believe, it may

System	# classes	Association	Composition	Generalization
RACE	9	5	2	1
Our system	10	10	2	2

Table A.1: Class diagram extraction comparison between RACE and our system.

result prejudicial due to the addition of unnecesary information. To compare between the different systems we have taken into the account the number of valid classes, associations, compositions and generalizations identified. We do not know how RACE obtained the attributes they included in their diagram, so we believe comparison between the two systems regarding the attribute validation is unrealistic.

Analyzing the results exposed in Table **??** we have observed that our system extracted an extra class "informadiko studentoj" (informatics students) and generalization relationship between that class and the class "studentoj" (students) that was ignored by RACE system. We also extracted the following extra five association relationships:

- Between "monpuno" (fine) and "fakultato" (faculty)
- Between "monpuno" and "librotenisto" (accountant)
- Between "libroj" (books) and "librotenisto"
- Between "biblioteko" (library) and "fakultato"
- Between "biblioteko" and "studentoj"

In the case of the associations identified only by our system, all are valid except the one between "monpuno" and "fakultato". The reason is that in the requirements is expressed that the fine should only to be charged to students. In summary, we identified an extra class, five more association relationships, which only one of them is amiss, one more generalization relationship, the same composition relationships and a dependency association when compared to the RACE system. Overall, we believe that with this test we proved that it is possible to succesfully apply natural language processing techniques with languages that are not extended. We eased the implementation when auxiliary systems are not available. It will impact on the manteinance and scalability of systems based in languages similar to Esperanto.

6.2 System vs experts

We gave the requirements document used by RACE, in English, to different experts and asked them to create a UML class diagram to the best of their knowledge. One of them is shown in Figure A.6.

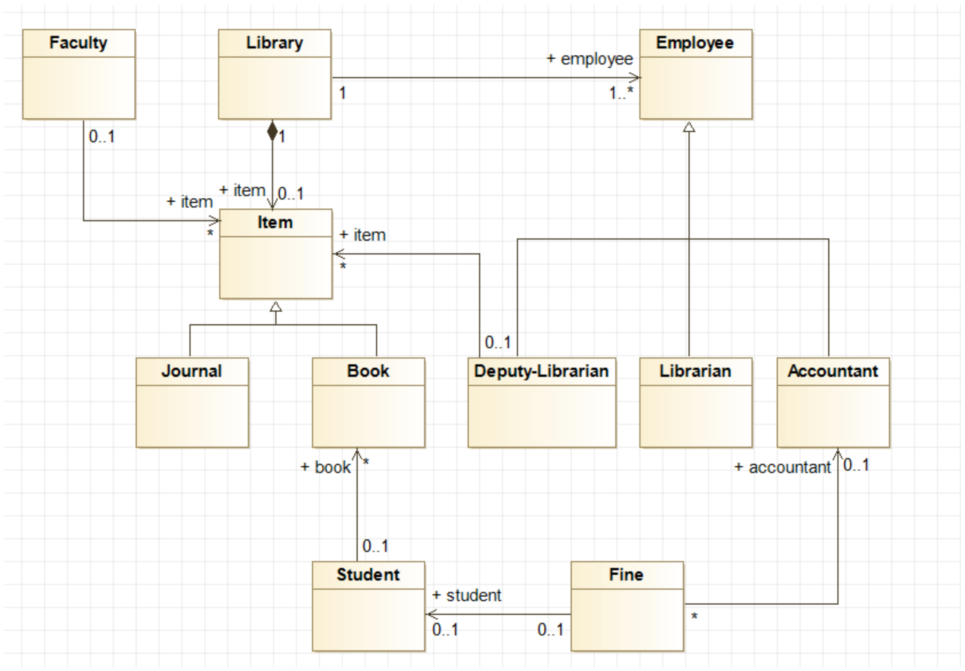
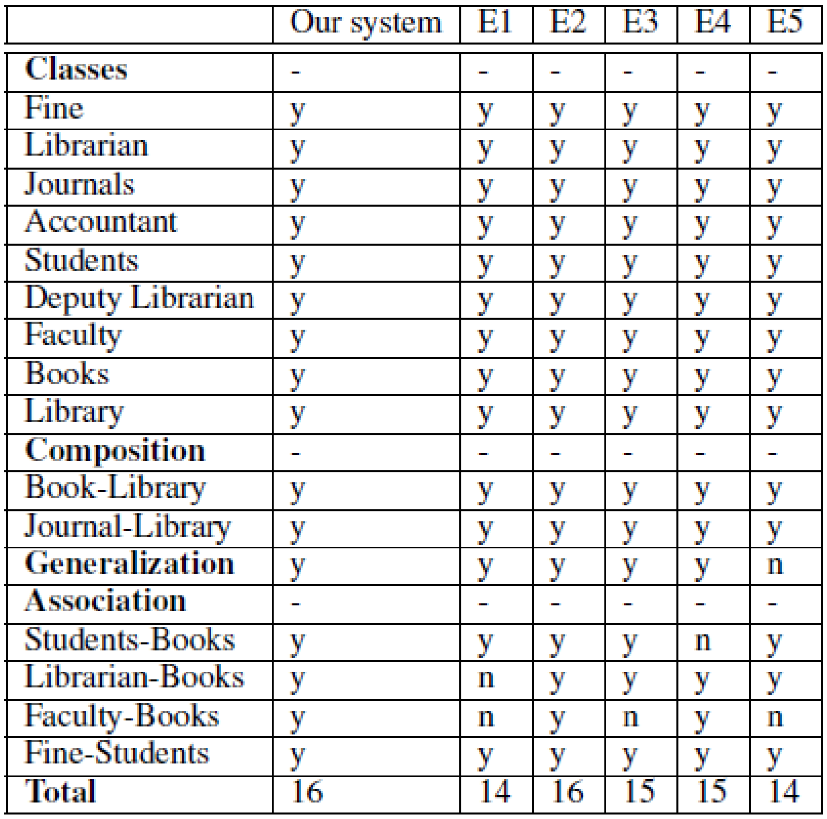


Fig. A.6: Class diagram extracted by an expert

This diagram is a more generic solution, looking at future additions in the system, like the two generic classes, "Item" and "Employee". Nonetheless, all the relevant classes and their relationships have been identified. To compare between the expert's diagram and our system's diagram, the following parameters will be analyzed:

- Whether all the relevant classes are identified or not. These classes being: "Fine", "Librarian", "Journals", "Accountant", "Students", "Deputy Librarian", "Faculty", "Books", "Library".



	Our system	E1	E2	E3	E4	E5
Classes	-	-	-	-	-	-
Fine	y	y	y	y	y	y
Librarian	y	y	y	y	y	y
Journals	y	y	y	y	y	y
Accountant	y	y	y	y	y	y
Students	y	y	y	y	y	y
Deputy Librarian	y	y	y	y	y	y
Faculty	y	y	y	y	y	y
Books	y	y	y	y	y	y
Library	y	y	y	y	y	y
Composition	-	-	-	-	-	-
Book-Library	y	y	y	y	y	y
Journal-Library	y	y	y	y	y	y
Generalization	y	y	y	y	y	n
Association	-	-	-	-	-	-
Students-Books	y	y	y	y	n	y
Librarian-Books	y	n	y	y	y	y
Faculty-Books	y	n	y	n	y	n
Fine-Students	y	y	y	y	y	y
Total	16	14	16	15	15	14

Table A.2: Class diagram extraction comparison between experts and our system.

- Check if the composition relationships "Book-Library" and "Journal-Library" exist.
- Check if A generalization relationship between "Librarian" and "Deputy Librarian" exist.
- Check if the following association relationships are present: "Students-Books","Librarian-Books","Faculty-Books" and "Fine-Students".

The results have been collected in Table **??**, showing that all the relevant classes that our system has identified have also been extracted by the experts, the composition relationships have been all extracted by our system and the experts, the generalization have been identified by the majority of both parties. Nevertheless, there are different results in the identification of the associations by the experts. We conclude this section by stating that our system has succesfully generated a class diagram with all the relevant classes implied and many of the association relationships between them, including also generalization relationships and composition relationships. In addition, dependency relationships have also been identified, and our system has been validated by comparison with an expert's diagram and a diagram created by RACE.

7. Conclusions and future work

UML class diagram generation using a requirements document expressed in natural language is challenging. It requires a fine grain approach in design and implementation. We have employed a domain specific language to model the behaviour of the Esperanto language in order to facilitate the natural language processing techniques that need to be applied to extract such diagram. We have used the most updated heuristic rules to identify concepts, classes, attributes and relationships. We have developed our prototype using Xtext, as we considered it is the most beneficial tool available at the moment which gives an exhaustive and detailed feedback to the user. We have performed two different evaluations: one comparing our system against another succesful system and another comparing the class diagram extracted by our system against those made by experts in the field. Our system could be improved in a few ways: for example by considering synonyms, so we don't add those elements that already exist in the diagram. The use of hyperonyms to better handle generalizations relationships and the heuristic rules used for relationships identification could be expanded to identify different relationships. Finally, other types of diagrams could be also developed using our system, such as use case or activity diagrams.

References

1 Philip Resnik et al. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res.(JAIR)*, 11:95–130, 1999.

2 John Bergey, Dennis Smith, Scott Tilley, Nelson Weiderman, and Steven Woods. Why reengineering projects fail. Technical report, DTIC Document, 1999.

3 Leon Starr and J Stephen. *Executable UML: how to build class models.* Prentice Hall PTR, 2001.

4 Grady Booch, J Rumbaugh, and I Jacobsen. The unified modeling language user guide addison-wesley. *ISBN: 0-201-571684*, 1999.

5 Booch Grady. Object-oriented analysis and design with applications. *Benjamin Cummings*, 1994.

6 Charles Richter. *Designing flexible object-oriented systems with UML*. New Riders Publishing, 1999.

7 Leszek A.. Maciaszek. *Requirements Analysis and System Design: Developing Information Systems with UML*. Addison-Wesley, 2001.

8 Mohd Ibrahim and Rodina Ahmad. Class diagram extraction from textual requirements using natural language processing (nlp) techniques. In *Computer Research and Development, 2010 Second International Conference on*, pages 200–204. IEEE, 2010.

9 Deva Kumar Deeptimahanti and Muhammad Ali Babar. An automated tool for generating uml models from natural language requirements. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 680–682. IEEE Computer Society, 2009.

10 Keletso J Letsholo, Liping Zhao, and Erol-Valeriu Chioasca. Tram: A tool for transforming textual requirements into analysis models. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 738–741. IEEE, 2013.

11 Xiaohua Zhou and Nan Zhou. Auto-generation of class diagram from free-text functional specifications and domain ontology. *Artificial Intelligence*, 2004.

12 Il-Yeol Song, Kurt Yano, Juan Trujillo, and Sergio Luján-Mora. A taxonomic class modeling methodology for object-oriented analysis. *Information Modeling Methods and Methodologies. Advanced Topics in Databases Series*, pages 216–240, 2004.

13 Bente Christine Aasgaard. Parsing of esperanto. Master's thesis, 2006.

14 Peter G Forster. *The Esperanto Movement*, volume 32. Walter de Gruyter, 1982.

15 Eckhard Bick. Tagging and parsing an artificial language. *proceedings of Corpus Linguistics 2007*, 2007.

16 GermaneSoftware. Homepage of EOparser, 2006. Accessed: 2017-04-04.

17 Eclipse Foundation. Xtext- Language Engineering Made Easy, 2006. Accessed: 2017-04-17.

18 Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

19 Kenn.Hussey. Getting Started with UML2, 2011. Accessed: 2017-05-17.

B. Acronyms

AST:	Abstract Syntax Tree
CIM:	Computation Independent Model
DNN:	Deep Neural Network
DSL:	Domain Specific Language
EMF:	Eclipse Modeling Framework
GMF:	Graphical Modeling Framework
GPL:	General Purpose Language
GUI:	Graphical User Interface
IT:	Information Technologies
JCR:	Journal Citation Report
MDA:	Model Driven Architecture
MDE:	Model Driven Engineering
NERHMM:	Name Entity Recognition using Hidden Markov Model
NLP:	Natural Language Processing
O-O:	Object-Oriented
OMG:	Object Management Group
PIM:	Platform Independent Model
POS:	Part Of Speech
PSM:	Platform Specific Model
RACE:	Requirements Analysis and Class diagram Extraction
SBD:	Sentence Boundary Detection

SENNA:	Semantic/syntactic Extraction using a Neural Network Architecture
SOM:	Semantic Object Model
TCM:	Taxonomic Class Modeling
TRAM:	Textual Requirements into Analysis Models
UI:	User Interface
UMGAR:	UML Model Generator from Analysis of Requirements
UML:	Unified Modeling Language
XML:	Extensible Markup Language

Bibliography

MaheshH.Dodani. A Picture is Worth a 1000 Words?, 2006. URL http://www.jot.fm/issues/issue_2006_03/column4/. Accessed: 2017-04-04.

Eclipse Foundation. Sirius- The easiest way to get your own Modeling Tool, 2007a. URL https://eclipse.org/sirius/. Accessed: 2017-04-17.

Kirill Fakhroutdinov. UML class diagram example for online shopping domain, 2013. URL https://goo.gl/1fPkvR. Accessed: 2017-04-25.

Philip Resnik et al. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res.(JAIR)*, 11:95–130, 1999.

Ian Byrd. Ambiguous Sentences, 2009. URL https://goo.gl/GBTjN4. Accessed: 2017-05-17.

John Bergey, Dennis Smith, Scott Tilley, Nelson Weiderman, and Steven Woods. Why reengineering projects fail. Technical report, DTIC Document, 1999.

Brad Matsugu. Poor Requirements, What impact do they have?, 2017. URL https://goo.gl/Uze9De. Accessed: 2017-05-17.

Leon Starr and J Stephen. *Executable UML: how to build class models.* Prentice Hall PTR, 2001.

Grady Booch, J Rumbaugh, and I Jacobsen. The unified modeling language user guide addison-wesley. *ISBN: 0-201-571684*, 1999.

Booch Grady. Object-oriented analysis and design with applications. *Benjamin Cummings*, 1994.

Charles Richter. *Designing flexible object-oriented systems with UML.* New Riders Publishing, 1999.

Leszek A.. Maciaszek. *Requirements Analysis and System Design: Developing Information Systems with UML.* Addison-Wesley, 2001.

Mohd Ibrahim and Rodina Ahmad. Class diagram extraction from textual requirements using natural language processing (nlp) techniques. In *Computer Research and Development, 2010 Second International Conference on*, pages 200–204. IEEE, 2010.

Deva Kumar Deeptimahanti and Muhammad Ali Babar. An automated tool for generating uml models from natural language requirements. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 680–682. IEEE Computer Society, 2009.

Keletso J Letsholo, Liping Zhao, and Erol-Valeriu Chioasca. Tram: A tool for transforming textual requirements into analysis models. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 738–741. IEEE, 2013.

Xiaohua Zhou and Nan Zhou. Auto-generation of class diagram from free-text functional specifications and domain ontology. *Artificial Intelligence*, 2004.

Il-Yeol Song, Kurt Yano, Juan Trujillo, and Sergio Luján-Mora. A taxonomic class modeling methodology for object-oriented analysis. *Information Modeling Methods and Methodologies. Advanced Topics in Databases Series*, pages 216–240, 2004.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

Sudha Morwal and Deepti Chopra. Nerhmm: A tool for named entity recognition based on hidden markov model. *International Journal on Natural Language Computing (IJNLC)*, 2:43–49, 2013.

Cícero Nogueira dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *ICML*, pages 1818–1826, 2014.

Dan Gillick. Sentence boundary detection and the problem with the us. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 241–244. Association for Computational Linguistics, 2009.

Bente Christine Aasgaard. Parsing of esperanto. Master's thesis, 2006.

Peter G Forster. *The Esperanto Movement*, volume 32. Walter de Gruyter, 1982.

Eckhard Bick. Tagging and parsing an artificial language. *proceedings of Corpus Linguistics 2007*, 2007.

GermaneSoftware. Homepage of EOparser, 2006. URL `http://www.germane-software.com/software/Utilities/EOParser`. Accessed: 2017-04-04.

Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

Anneke G Kleppe, Jos Warmer, Wim Bast, and MDA Explained. The model driven architecture: practice and promise, 2003.

Joaquin Miller, Jishnu Mukerji, et al. Mda guide version 1.0. 1, 2003.

JLG Dietz and JAP Hoogervorst. Enterprise ontology and enterprise architecture–how to let them evolve into effective complementary notions. *GEAO Journal of Enterprise Architecture*, 2(1):121–149, 2007.

Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

Arie Van Deursen, Paul Klint, Joost Visser, et al. Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6):26–36, 2000.

Eclipse Foundation. Xtext- Language Engineering Made Easy, 2006. URL `https://eclipse.org/Xtext/`. Accessed: 2017-04-17.

Vladimir Viyović, Mirjam Maksimović, and Branko Perisić. Sirius: A rapid development of dsm graphical editor. In *Intelligent Engineering Systems (INES), 2014 18th International Conference on*, pages 233–238. IEEE, 2014.

Moritz Eysholdt and Heiko Behrens. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 307–309. ACM, 2010.

Heiko Behrens. Mdsd for the iphone: developing a domain-specific language and ide tooling to produce real world applications for mobile devices. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 123–128. ACM, 2010.

Henning Heitkötter, Tim A Majchrzak, and Herbert Kuchen. Cross-platform model-driven development of mobile applications with md 2. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 526–533. ACM, 2013.

Jose M Conejero, Juan Hernandez, Pedro J Clemente, Roberto R Echeverria, Juan C Preciado, and Fernando S Figueroa. Automatic configuration of video-surveillance applications: a model-driven experience. *IEEE Latin America Transactions*, 13(8):2700–2708, 2015.

Mathias Funk and Matthias Rauterberg. Pulp scription: a dsl for mobile html5 game applications. In *International Conference on Entertainment Computing*, pages 504–510. Springer, 2012.

Johan den Haan. DSL and MDE, necesarry assets for Model-Driven approaches , 2009. URL `https://goo.gl/UKlRaS`. Accessed: 2017-04-25.

Kenn.Hussey. Getting Started with UML2, 2011. URL `https://goo.gl/VJyxvy`. Accessed: 2017-05-17.

Eclipse Foundation. Papyrus, 2007b. URL `http://www.eclipse.org/papyrus/`. Accessed: 2017-04-17.