Departament de Llenguatges i Sistemes Informatics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Master in Computing

Master of Science Thesis

Integration of Multidimensional and ETL design

Petar Jovanovic

Advisor/s: Dr. Alberto Abelló Gamazo
Dr. Oscar Romero Moral

June, 23rd 2011.

Acknowledgements

These lines I want to dedicate for giving my thanks to all people without who this thesis would have not beeen possible.

I want to thank my thesis advisors, professors Alberto Abello and Oscar Romero, first for believing in me and for giving me the opportunity to work with them on this novel approach, and then for being a great help during my work, with their constant patience, useful advices and suggestions. I am especially grateful to them for encouraging me to think wider about the problems and to delve deeper to find the right and quality solutions.

I also thank professor Oscar Romero and Daniel Gonzalez for cooperation and great help during the integration process of the GEM system.

My thanks also go to my friends who, during the rough moments, cheered me up and gave me the strength to go on.

However, my deepest gratitude I owe to my parents who gave me the chance to even be at this place and who supported me the most through all my life and education. I hope I will give them the reason to be proud of me. My special thanks go to my sister for always being there to encourage me and to make me believe in myself and my work.

Index

Index of figures

Index of tables

1. Introduction

Modern civilization is characterized by a constant need to follow, collect and store data about various events. Information systems have very easily become present in all the areas of the human lives, while databases that support them have enlarged to the scale of even petabytes. Billions of records in those databases do not necessarily need to be just a note that something specific happened in the past. They can be modeled and shaped in the way to represent meaningful pieces, that can be used, based on many recorded data, to infer some new knowledge, to follow the pattern of some changes and finally to help people in business to make some important decisions.

The time when business people were struggling with huge amount of data which did not have any useful meaning is almost passed, thanks to Business Intelligence which has recently become one of the most promising areas of IT world. Today and even more in the future, the companies would not be able to compete in the world market if they do not provide an intelligent way of analyzing their data and extracting information that is crucial for their revenue growth.

Decision-making support systems represent the subclass of BI information systems. According to transactional sources used in everyday business, and additional usage of business specific logic, these systems should be able to identify incurred problems and to propose corresponding solutions.

These systems, depending on the business needs, tend to become very complex. The beginning, but the crucial task of every project is appropriate design of the corresponding data warehousing system.

Data warehousing systems are aimed at exploiting the organization data, previously integrated in a huge repository of data (the *data warehouse*), to extract relevant knowledge of the organization. As formally defined in [19], *Data Warehousing* represents a collection of methods, techniques and tools used to support knowledge workers, i.e., senior managers, directors, managers and analysts, to conduct data analyses that help with performing decision-making processes and improving information resources.

The first introduction to *Data Warehouse* concept dates back to 1992 and its first definition is given by B. Inmon. *"Data Warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process".* This actually means that the Data Warehouse represents a single storage for all domain-relevant data, from various available sources (*integrated*), collected during the particular period of time (*time-variant*). Additionally, this also states that this storage is stable in terms that data can only be inserted but never updated or deleted form data warehouse (*non-volatile*). Even though the area of decision making systems evolved a lot, this definition is still mostly accurate.

As it is stated, the data warehouse is a huge repository of data, but it does not tell much by itself. Therefore, the tools for extracting the information that can be useful in a decision-making process should be introduced. These tools are known as the exploitation tools. The ones that are the most relevant to my work in this thesis are the *OLAP* tools (*On-Line Analytical Processing*). The OLAP's main objective is to analyze business data from its dimensional perspective. These tools are generally conceived to exploit the data warehouse for analysis tasks based on *multidimensionality*.

Multidimensionality represents a paradigm based on the *fact/dimension* dichotomy and it is intended for representing data as if placed in an n-dimensional space, which allows easy understanding and analysis of data in terms of *facts* (the subjects of analysis) and *dimensions* showing the different points of view from where a subject can be analyzed.

The fact that OLAP tools are used for analysis based on multidimensionality, arose the necessity for the appropriate *multidimensional design (MD)* of the underlying data warehouse.

At this point, for the sake of better understandability, the specific multidimensional modeling terminology from the above mentioned fact/dimension dichotomy point of view, and that is used through this thesis, is introduced.

- ***Dimensional concepts*** represent the desired perspective, i.e., part of multidimensional space, from which we observe the fact (e.g. Time, Place). This perspective can be defined by:
    - ***dimensions*** that can contain a hierarchy of ***levels*** representing different granularities of data(e.g. Time: Year ; Place: City), and
    - ***descriptors*** (**slicers**), which are the means for describing the characteristics and context of a particular level (e.g. Year = "2006", City = "Tokyo").
- ***Factual data*** containing ***measures*** represents the data of interest for the analysis process (e.g. number of sold products). Value of this data is obtained according to chosen dimensional concepts (perspective).

Besides the appropriate multidimensional design (MD) of the underlying data warehouse and exploiting (OLAP) tool, data warehousing systems requires the means for managing the dataflow from the available sources to the target MD schema. This supporting process is called the *ETL process* and it has the main role in *Extracting* data from the chosen data sources, appropriately *Transforming* and homogenizing those data and *Loading* them into the underlying storage (data warehouse).

Since in all IT projects business requirements are often a precursor to designing and building a new business application/system, the design of the data warehousing systems also highly depends on the business needs expressed as requirements. Therefore, a very important task is to fully understand requirements coming from the business specialist and to correctly transform those requirements into the appropriate MD and ETL designs.

1.1. *Starting point*

Due to high complexity of MD and ETL structures, correlated with the lack of understanding business needs, not that rarely decision-making support projects fail.

Professors Alberto Abelló and Oscar Romero from ESSI department (Departament d'Enginyeria de Serveis i Sistemes d'Informació) at Technical University of Catalonia (UPC), together with Alkis Simitsis have recently presented, in [11], the GEM framework. GEM represents the system that semi-automatically produces both the data warehouse multidimensional (MD) design and the ETL process designs of the resulting data warehousing system, concerning both the set of business requirements and source data stores as the inputs. This system tends to support designers during the early stages of the DW design projects, by offering them help in overcoming obstacles that have previously threaten to hold down the whole project.

During past several years, the architecture of the GEM system, presented recently in [11], has been fully developed. (Figure 1)

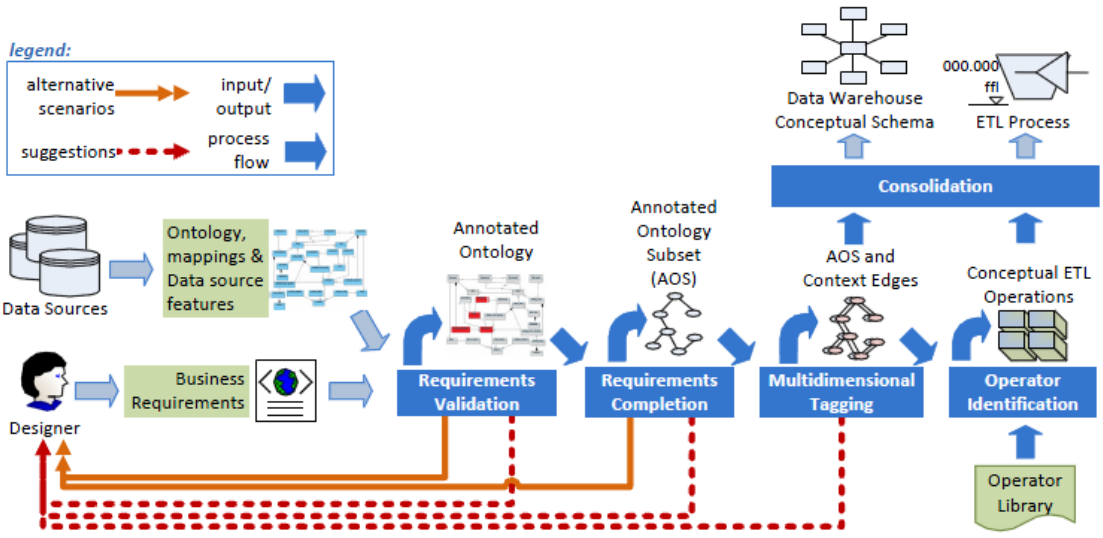


Figure 1: GEM framework architecture

The GEM framework, during the process of producing the ETL and MD conceptual designs, passes five different stages. In the sequel, only brief introduction to these stages will be provided.

Nowadays, it is well known that any data warehousing system must treat as first-class citizen both the business (analytical) requirements and the source data that will populate the target data warehouse. Therefore, the GEM system, developed in this approach, takes as an input three different pieces of information:

- **Source data stores** whose semantics, characteristics and constrains are represented in a tailored ontology.

- **Mappings** that are expressed as an XML structure and that represent the information if an ontology concept is mapped to the real source data and how it is actually mapped.
- **Business requirements** are also expressed in a structured form of an XML file, which represents the analytical query needed for the organizational decision making process.

It can be noticed that the first two pieces of information represent the information about the data sources, while the third one represents the requirements coming from the business specialists.

Starting from the concepts stated inside the input business requirements, the first stage, **Requirements Validation,** searches for the corresponding concepts in the sources, i.e., in the ontology. After the required concepts are identified in the ontology, they are then tagged according to the multidimensional role they may play (Level, Measure or Descriptor). Afterwards, the system searches for mapping of each tagged concept to the source data stores. The set of ontology concepts identified from the business requirements, tagged and mapped in the Requirements Validation stage, is then complemented with the additional information from the sources, during the following stage of **Requirements Completion**. This stage identifies intermediate concepts that are not explicitly required in the input requirements, but are needed in order to retrieve data that will fully answer the input requirements. These concepts are later also tagged with their appropriate multidimensional role. Afterwards, the stage of **Multidimensional Validation,** validates the correctness of these taggings, as a whole, according to multidimensional design principles. The following stage of GEM, **Operation Identification**, identifies the ETL operations needed to support the dataflow from the source data stores to the newly created MD target data stores. ETL operations are here, using the meta-knowledge generated in the previous stages, identified in three separated phases. The final stage of the GEM is **Conciliation stage**. All the above stages should be run once for each input business requirement. The designs produced for each business requirement are then conciliated, during this stage, into a single multidimensional design and single supporting ETL process design.
Besides the architecture, different stages of the framework have already been studied in depth and developed. The stage of Multidimensional Validation has been fully developed by Professor Oscar Romero in [12], within his PhD thesis, while the Operation Identification stage has been implemented by Daniel Gil Gonzalez, as part of his final university project at Technical University of Catatonia (UPC).

The basis of my master thesis represents the initial phase of the GEM framework. This phase, in fact, includes integration of the elicited business requirements, with the whole process of automation of the MD and ETL conceptual designs. It consists of building a process that would interpret the input business requirements according to the available sources, and translate them into structures that the final stages of GEM require as their inputs. This actually intends to automate the manipulation of business requirements and integrates them in the whole

framework. This process is actually covered with the first two stages of the GEM framework. During the stage of Requirement Validation, this process, as already explained, first validates input requirements, tags them with their multidimensional roles, and maps them to the sources. At the same time, during this stage, based on these mappings, the process builds the initial ETL structure. This structure represents the part of the input of the Operation Identification stage and actually includes the set of the ETL operations needed to extract the required concepts form the source data stores. Afterwards, during the Requirement Completion stage, the process searches the ontology and tries to relate required concepts inside the data sources, and as a result it produces a graph structure. This structure contains identified paths between tagged concepts including also the intermediate concepts and associations found on those paths. Finally, this graph structure represents the input for the Multidimensional Validation stage while it is also essential for the identification of the operations of the output ETL process design.

1.2. Motivation and Objectives

Business requirements are usually collected in various, sometimes even unstructured ways. Questionnaires, checklists, surveys, recorded oral conversations etc. Even when we deal with the structured forms of the requirements, process of translating these requirements into the desired design requires high effort and is usually error-prone.

As nowadays the markets are highly competitive and have strong global tendency, corporations has to make rapid and smart decisions in order to survive on it. Therefore, companies' revenue growth highly depends on the existence and quality of the decision-making systems.

This project is mainly motivated by the desire to eliminate mentioned obstacles in the early stage of the multidimensional and ETL design, and to possibly lower the current projects failure rate. It can be stated here, that the main goal of my work is to provide the GEM framework with the pre-process that will higher the level of automation of the whole system and generalize the system's input data formats.

1.3. Scope of the project

This project represents master thesis and the final project, on the Master in Computing program, at Technical University of Catalonia.

Led by the motivations and goals previously expressed, this project consists of the following:

- *Theoretical part*. This part represents the research in the field of automating and customization of multidimensional and ETL designs. It also includes exploration of the previous attempts in building a system which would lead system designers during the process of the ETL design, and

- *Technological part.* This part includes the realization of the initial stages of the GEM framework. Besides implementation of these stages, technological part of the thesis also includes complete integration of the initial stages with the other, already implemented stages of GEM, i.e., Multidimesional Validation (MDBE) and Operation Identification (ETL generation), into the whole framework. These stages have been developed by professor Oscar Romero and Daniel Gil Gonzalez respectively.

1.4. Structure of the document

As this document is the final university project, it is structured in the way to completely represent my work during the project, beginning with the research and theoretical part, followed by the complete documentation of the development and testing processes.

Chapter 2 represents the State of the Art in the field of automation of ETL process design including some early customization attempts and some newly approaches based on ontologies. In the sequel, chapter 3 exhaustively talks about new approach, Requirement-driven generation of ETL and Multidimensional Conceptual Designs, which is the basis of this project. Chapter 4 includes complete technological work of this project. First there is the discussion about the development method that is applied in this project. Afterwards, the processes of design, implementation and integration of the GEM framework are explained in detail. At the end of chapter 4 the testing process, of both my part of the framework and the whole integrated framework, is discussed. In chapter 5, there is a short discussion related to the time that is planned and actually spent on this project and accordingly the project costs are estimated. At the end, in chapter 6, some conclusions and the possibilities for the future work are provided. Additionally, the short demo of the developed application and the user manual are represented in Appendix A.

2. The State of the Art

This chapter represents the beginning of theoretical part of the thesis and includes overview of the work of most prominent researchers in the field of automation and customization of the multidimensional and ETL design.

As already mentioned, GEM framework facilitates both production of multidimensional and ETL designs. It can be stated here, that, to our best knowledge, GEM represents the first approach, which synchronously produces conceptual design of both target conceptual schema and supporting ETL process. Indeed, GEM represents two systems integrated in one framework.

Concerning multidimensional design, a lot of work, within this research group, has been done. Starting from the research of professor Oscar Romero, during his PhD [12], which also includes implementation of the Multidimensional Validation stage of the GEM framework (MDBE). An exhaustive survey has also been conducted, by professors Oscar Romero and Alberto Abelló in [13]. Here, they comment the relevance that data warehousing systems have gained, for supporting decision making, in the last years. The main objects of their survey are the methods, developed during the last decade to support data warehousing. After discussing common terminology and notation, the authors in [13], chronologically present current multidimensional design methods in terms of the stated terminology. They start from the earliest attempts in 1998, until the most recent works in 2010. Afterwards, they also introduce comparison of the presented methods based on the criteria that they iteratively introduced during their research.

Since the exhaustive research in the field of multidimensional design has been done within this research group, this state of the art only considers the topic of ETL design, with the focus on the approaches that tries to automate or generalize and customize ETL design.

Until now, various technologies concerning the design of the ETL process have been released. However, even with these existing technologies, the human-conducted design of ETL is very demanding and error prone. In order to lower the effort and risks involved, various researchers have analyzed the possibility to partly automate the process of ETL design. In the survey that Panos Vassiliadis conducted in [14], both conceptual and logical modeling of ETL process, are covered. At the same time, the survey concerns each stage of the process (Extraction, Transformation and Load) and discusses problems related to the ETL process. The author begins with the first approach towards conceptual UML–based modeling of data warehouse, with special focus on the back stage of data warehouse, the ETL process. This approach, Stohr, Muller and Rahm (1999), from the end of the previous century, represents nearly the first attempt for customization of ETL process. It is stated that the framework that authors provided allows customization of ETL activities and easy plugging of new ones, through some kind of specialization. In the sequel, author presents his own work

(Vassiliadis et al, DOLAP 2002) based on ad-hoc model and motivated by the fact that the designers during the early stage of the data warehouse modeling are mostly concerned with the analysis and content of the existing data sources and their mapping to the new target model. The author then presents the approach that revisits the UML idea for conceptual modeling of ETL workflows, Trujillo & Luján-Mora (2003). The authors of this approach express doubts about the previous (Vassiliadis et al) approach, mostly because it was based on an ad-hoc structure. Therefore, Trujillo & Luján-Mora (2003) in their work employed standard methods (UML) for conceptual modeling of the ETL. Interestingly the authors employed class diagrams for modeling the ETL workflow, with the main reason that the focus of the modeling is on the interconnection of activities and not on the actual sequence of activity steps. As a result, Trujillo & Luján-Mora (2003), provide generic structure for the design process for ETL workflow and the six-stage process for building this generic structure. The author of the survey also covers the semantic-aware design methods for ETL. In this section various approaches are briefly presented. First, the approaches towards the semi-automatic transition from the conceptual to the logical model for ETL process (Simitis(2005) and Simitis&Vassiliadis(DSS 2008)) are presented. In the sequel, the author shortly mentions the ontology based approaches [2,5].

Following the work of this survey, concerning some additional ETL customization approaches and fully covering some ontology based that are only briefly introduced in the survey, this state of the art specially focuses on four approaches [1,2,3,4]. These approaches are found as the most relevant, regarding the topic of this work, based both on the number of their citations and the relevance of their authors. Some of these works focus on the generalization and customization of ETL design while others mostly focus on ontology based methods towards the automation of ETL design.

2.1. Introduction

The crucial parts of decision-making system projects are, without doubt, constructing the data warehouse and supporting the ETL process. Conceptually, data warehouses are used for the timely translation of enterprise data into information useful for analytical purposes [1]. This information is collected from various (mostly relational) sources, transformed to satisfy analytical needs and DW constraints and loaded into DW, by the ETL process.

It has been stated that the main reasons for earlier mentioned failures of decision-making projects are the inherent amount of complexity of these environments and the technical details into which the designer must delve, in order to deliver the final design. Such complexity is mainly caused by the fact that the designer has to choose from the existing sources, those pieces of data that will be used and loaded in the target DW and additionally and even more difficult to identify, the transformations that should be applied to selected data before loading them to the DW. As these projects are financially and timely very expensive, a common understanding between all involved parts is also very important. As stated in [1], approximately 30–50% of the data warehouse design process is spent on analysis, to ensure that source systems are understood

and that there is alignment between the involved parties: business and administrators. For reaching this common understanding it is of great importance to provide both a formal (explicit) and more natural (human readable) way to represent all the parameters and properties guiding early stage of DW design.

2.2. Objectives

This overview represents the starting point in the theoretical part of my work and has main goal to identify and examine the previous approaches and solutions that use generalization and partial automation of ETL design to lower efforts, costs and risks involved in projects of implementing decision-making support system.

2.3. Content and structure

First approaches tried to deal with providing general and customizable ETL scenarios that can be adopted in various cases and environments [3, 4]. Several approaches have been proposed for using Semantic Web technology to the design and construction of the conceptual part of the ETL process. [1,2] Some researchers deal with one of the major challenges of the ETL design: the structural and semantic heterogeneity. The core idea of their work is the use of ontologies to formally and explicitly specify the semantics of the data source and the data warehouse.

In the following sections all the above approaches will be examined in more details and additionally some conclusions will be presented. In section 2.4, the proposal of an ETL tool, from the beginning of the century that deals with the above mentioned issues of generality and customization, will be presented. In the later section, new ontology-based approaches will be covered. At the end, the conclusion about current state of art in the domain of ETL and DW design will be provided.

2.4. Early attempts

First approaches [3,4] were mostly focused on providing general and customizable solution, additionally with the sets of frequently used ETL scenarios that can easy the job of the designer and lower the risks of the whole project.

Even though, these works [3,4] are not so fresh, they should be considered in this overview, because of their historical relevance as one of the first attempts to deal with the problem of complexity and reusability of DW and ETL.

In [3], it is stated that the majority of problems related to the Data Warehouse and ETL implementation comes from the complexity of these concepts and they mentioned two main reasons of such complexity:

1. *Technological reasons* – due to the fact that DW is a heterogeneous environment where data must be integrated both at the *schema level (*e.g., *naming conflict)* and at the *instance* level
2. *Qualitative reasons* – data warehouse, as a part of the decision support system, must provide high-level quality of data and services. Accordingly, it is noticed in [3], that data quality problems seem to introduce even more complexity and computational burden to the loading of the data warehouse

During their research, in [3], the authors found out, that even though there are various commercial ETL tools already presented in the market, the companies rarely decide to use them. This is, beside performance issues, mostly because of their ubiquitous complexity and because of the time and resources that they need to invest for learning them. As a result, they noticed that the companies usually spent one third of their data warehouse budget to the ETL tool that is, in most of the cases, built in-house.

For dealing with generality, customization and reusability the authors in [3, 4] developed a tool (Arktos and in [4] Arktos II). The basis of their tool is metamodel primary developed in [3] and later improved in [4].

This metamodel contains three layers:

- Metamodel layer
- Specialization (in [4] Template) layer
- Metadata (in [4] Schema) layer

The first layer of the metamodel (Metamodel layer) is composed of generic entities and accordingly is supposed to provide *generality*, with the derivation of the simple model that is powerful to capture luckily any ETL scenario.

To deal with the second issue of reusability and customization, the authors included the second layer (Specialization and in [4] Template layer). This layer should be used to add the most frequently used ETL activities. The Template layer is actually represented as a specialization of the entities from the Metamodel layer. With providing templates, this part of tool is supposed to be used to ease the future starting faze of the ETL process development. In [4], the authors completely and formally explained the procedure of adding new templates which can be very helpful for future projects –"Customization mechanism".

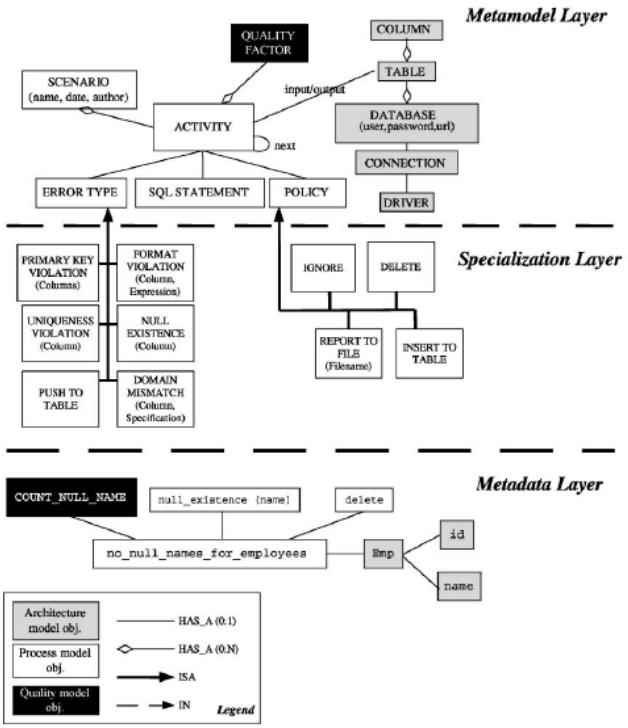


Figure 2: Metamodel from [3]

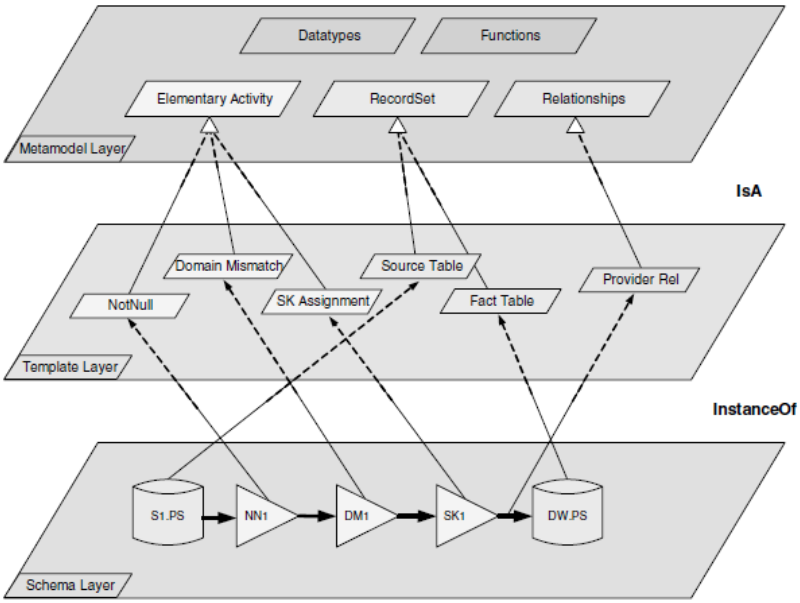


Figure 3: Metamodel from [4]

This mechanism should indeed be used by designers at the end of one project, to examine which ETL activities they frequently used during the project and to give them the possibility to enrich the set of template activities for the future projects. In the sequel, one can see the table of frequently used template activities that authors of [4] originally provided.

Filters	Unary operations	Binary operations
- Selection (σ) - Not null (NN) - Primary key violation (PK) - Foreign key violation (FK) - Unique value (UN) - Domain mismatch (DM)	- Push - Aggregation (γ) - Projection (π) - Function application (f) - Surrogate key assignment (SK) - Tuple normalization (N) - Tuple denormalization (DN) **File operations** - EBCDIC to ASCII conversion (EB2AS) - Sort file (Sort)	- Union (U) - Join (⋈) - Diff (Δ) - Update Detection (ΔUPD) **Transfer operations** - Ftp (FTP) - Compress/Decompress (Z/dZ) - Encrypt/Decrypt (Cr/dCr)

Figure 4: Template activities originally provided in [4]

The tool provided in [3] (Arktos) and later improved in [4] (Arktos II) covers three different ways for describing scenarios. Beside graphical interface they offered two declarative languages XADL, which is an XML variant that provide more verbose way and SADL, which is SQL-like language and thus with more compact syntax. The authors also provided comprehensive explanation of these languages.

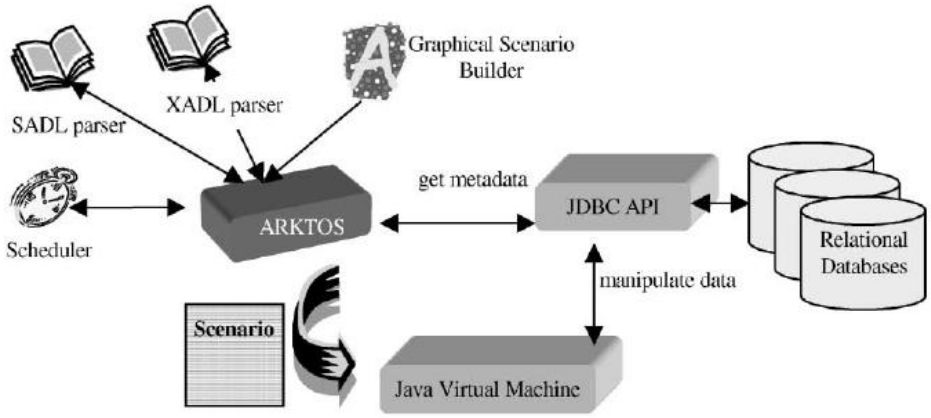


Figure 5: The architecture of Arktos, from [3]

The authors in [4] supported their work, mentioning the reasons why the designers would be encouraged to use their tool. They stated that Arktos II overcome the issues of complexity and amount of effort needed for learning. Their tool provides assistance to the designers by offering following functionalities:

- Friendly GUI with all the features available through the forms and point-and-click operations, and
- Set of reusability templates

Their solution also covers the issues of:

- *generality* by the fact that any activity can be additionally tailored by the user (provided by metamodel) and
- *customization* by the fact that the tool on one side offers already created activities and on the other side the opportunity to users to customize the environment according to their needs

As previously stated, the quality issue, that is usually obliged in the decision support projects, brings a certain level of complexity. The authors in [4], additionally provided a feature, inside of Arktos II, that deals with this issue. This feature provides computation of scenario's design quality by employing the set of metrics, and it can be applied both to the whole scenario and to individual activities.

2.5. Ontology-based approaches

As it has already been stated, the main problem with decision support projects is inevitable complexity. The authors in [2], point out the main reasons for the abovementioned complexity.

First one is that the ETL design is mostly driven by the semantics of the data sources on the one side, and constraints and requirements of the DW on the other side. Here the problem is that this information is usually available in natural language in the form of specification, documentation, comments, oral communications etc.

Second, but not less serious and also the main challenge in the back stage of DW is heterogeneity among the existing sources. According to [2] two main heterogeneities are present:

- structural heterogeneity that refers to the fact that in different data stores the same information is stored in different structures, and
- semantic heterogeneity with three conflicts that cause this kind of heterogeneity
    1. *confounding conflicts*, which occur when information items seem to have the same meaning, but differ in reality; e.g., owing to different temporal contexts;
    2. *scaling conflicts*, which occur when different reference systems are used to measure a value; e.g., different currencies or different date formats, and
    3. *naming conflicts,* which occur when naming schemes of information differ significantly (a frequent phenomenon is the presence of homonyms and synonyms.)

The authors in [2], deal with these kinds of heterogeneity, and propose an approach that facilitates the construction of the ETL workflow at the conceptual level.

For their solution, they chose ontology-based approach and they commented some of the reasons for that:
Ontology:

- Allows *formal* (machine readable) way of definitions,
- Supports *explicit specification* of data sources and DW, and
- Provide *well-defined semantics* that would be convenient for desired automation of ETL process

For representing these ontologies they used Web Ontology Language (OWL), because OWL provides a formal representation and the ability to leverage on existing reasoners for automating several tasks of the process, such as checking subsumption and transitivity relationships between ontology elements. Another reason is also the fact that OWL is the proposed W3C standard for representing ontologies on the Web. OWL is known as very complex, but authors stated that only a subset of OWL is used in their approach.
Their solution consists of four different steps, and these are depicted in figure 6.

This approach, [2] is relevant and thus, included in this overview mostly because it is focusing on automation of identification of the ETL transformations required for ETL process by using ontologies for formal and explicit specification of the semantics of the data stores' schemas.
It is shown in [2] on the simple example, that having this formal and explicit description of the domain, it is possible to automate in a large degree the process of the ETL workflow creation and to overcome problems of complexity caused by heterogeneity.

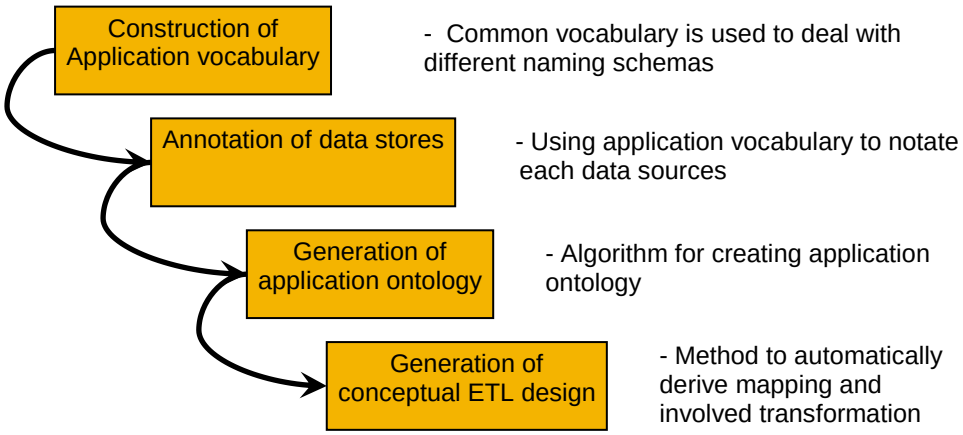


Figure 6: Process of generation of conceptual ETL design

In the previous work it is shown that ontologies are suitable for capturing the information needed for the generation of conceptual design of ETL processes. This approach, in the large amount, eases the task of identifying the appropriate mappings and the necessary transformations, by the fact that each data store is semantically annotated in the domain ontology. Having the ontology annotating the data stores, the usage of reasoner simplifies and automates the task of identification of correspondences between the sources and the data warehouse.

As much as this solution offers explicit and concise output, it is expressed in formal language, which creates new challenge and additionally complicates the communication among the different parties and also makes the validation of the result harder for the designer.

The same authors from [2], supported by Malu Castellanos, considered this problem and in [1] proposed the solution that also uses semantic web for automating ETL design, but with the additional focus on human readable output.

In [2], the authors tackle this problem by exploiting the fact that the ontology contains all the appropriate metadata describing the ETL process. Based on that fact, they tried to develop a method to translate the result of the reasoner into a narrative form, which represents the most natural means of communication. Narrative description of the generated conceptual ETL operations makes it easier for the involved parties to validate the design and to generate reports and documentation regarding the involved activities.

In figure 7, the architecture of the proposal from [1] is represented. First two phases of this approach are mostly those from the previous solution [2], so it will not be further discussed.

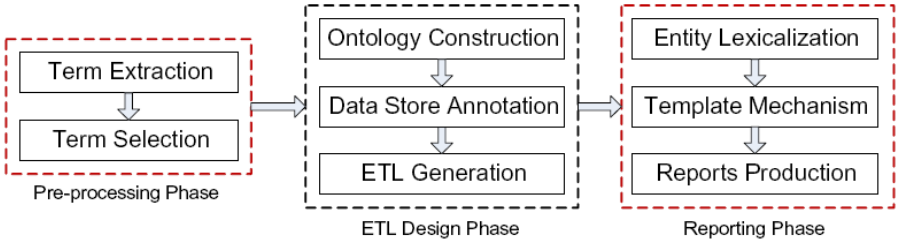


Figure 7: Architecture of the model, from [1]

The main focus will be on the *Reporting Phase* with its main goal of generating reports regarding the ETL process and the involved data stores, in a natural language format.

A list of data store annotations and ETL transformations, produced in the first two phases are the input of the Reporting Phase. The process of *lexicalization* (of entities) is introduced in [1] as the first step in this phase. This term refers to the process of deciding which words or phrases to be used for transforming an underlying message into a readable text i.e., which word or phrase to associate to an entity that is defined as a class or property in ontology.

Besides *lexicalization*, a *template mechanism* is stated to be the second step of this phase. Its main role is to generate the reports, using predefined templates in order to represent the design constructs in a textual description.

The output of this phase is textual description in natural language and thus readable by the various parties involved in the project.

In both previous cases the semantic web approach (ontology-based) is used, to explicitly and formally define the data stores for future use by the resoner and additionally to formally define

ETL transformations. The last approach additionally deals with the problem of translating this formal definition of ETL conceptual model into the human readable form.

2.6. Conclusion

As it is stated in this overview, there are several problems that occur during the development of Decision support systems. These projects are known as very complex and error-prone and thus, time-consuming. In order to overcome obstacles that occur during the project the idea of automation of the ETL design arose.

This overview covers several approaches from the beginning of the century, until now. First approaches [3,4] were initially concentrated on the design of ETL process and they proposed a tool (Arktos and Arktos II). Their solution on the one hand provides certain amount of generality and thus tries to cover all possible ETL scenarios, and on the other hand offers some of the most commonly used ETL activities for users to have it as a kind of templates for future projects. Later works [1,2] based their solutions on the usage of semantic-web (ontology-based) technologies to annotate the data stores and ETL transformation, which will be suitable for use by the reasoner for establishing the connections between source and target stores and supported ETL activities.

The possible problem with subjectivity, that one may find in this overview, as Alkis Simitsis appears as one of the authors in most of the covered papers, can be explained by the fact that he is one of the most important researchers in this field and additionally by the fact that through the years he worked with various prominent researchers that certainly must be included when we talk about ETL conceptual design.

From the presented overview it can be concluded that the attempts for the automation of ETL design evolved in a large amount. A lot of efforts have been invested in generalization and customization of the ETL process. All these efforts had the same aim - to help designers and remove obstacles from the early stages of the decision-support projects. However, none of these works specifically considers business requirements and the amount in which these requirements drive the ETL design. On the contrary, GEM framework, along with the available sources, takes input business requirements, as the basis of the both ETL and MD designs. At the same time, unlike existing approaches, GEM additionally considers the synchronous generation of target MD data stores.

3. New approach - Requirement-driven Generation of ETL and Multidimensional Conceptual Designs

After the overview of the existing approaches in the previous chapter, the GEM framework, the system that is briefly introduced in section 1.1., and that is the basis of this thesis, is covered in more details by this chapter.

As already mentioned, one of the main objectives during the initial phase of every IT project, is a complete understanding of the business requirements that have to be fulfilled by the end. The same principle holds in the decision-making projects including its crucial part of designing the data warehousing system. During these projects, business requirements should be gathered and fully understood, before they are translated to the appropriate designs. That is why this process often includes several rounds of reconciliation and redesigning, before it is certain that all the requirements are fully satisfied.

There are usually two distinct parties involved in the process of requirements gathering: the business analysts and financial experts on the one side, and technical designers, IT experts, on the other side, whose aim is to translate the needs of the business people into the appropriate designs.

Unlike the previous approaches, which considered that target multidimensional schema already exists, the GEM framework aims to provide the means for semi-automatic generation of both Multidimensional and ETL conceptual designs. The method used in this approach combines information about the data sources with the gathered business requirements. Then it validates and completes these requirements and produces multidimensional design and supporting ETL operations simultaneously.

3.1. GEM Framework

The GEM Framework is briefly introduced in section 1.1. The inputs that GEM framework expects along with the framework stages, which are necessary for handling these inputs and that are actually the part of my thesis are explained in more details in the following sections. Final stages of the GEM framework that are already implemented (Multidimensional Validation and Operation Identification) are briefly introduced here for fully understanding of the GEM's big picture and for understanding how the initial stages bridges the gap from general input formats.

3.1.1. GEM Inputs

3.1.1.1. Source data stores

This part of the input represents the structure of the sources from which the data will be extracted, transform and finally loaded into the target schema. Semantics and constraints of the available data sources, in the GEM system, are captured in terms of an ontology. As ontology language, the OWL is used. More details about OWL and reasons for using it are presented in section 4.2.5., prior the implementation part. It is explained in [5], that both structured and unstructured data stores can be represented in the form of graphs and thus consequently in the appropriate ontology. It is also stated that the process of construction of the ontology, based on the source data stores, is mostly led by the integration of a business domain vocabulary with data sources' vocabulary. More details about the ontology construction process, based on data stores, are provided in [5], while in this work it will be considered that the constructed ontology is already available at the input.

Even though the OWL ontology is capable to represent various types of data stores, for this thesis, the examples of representing relational data sources are provided (e.g., tables, attributes, cardinalities etc.). These examples are based on the TPC-H benchmark, which is used for the testing process and explained in section 4.4. The complete ontology for the TPC-H data sources is depicted in Figure 28 in Appendix A, where it is used for representing the demo of the developed system.

The part of the TPC-H schema covered with these examples contains the following information:

- Table *Region* with the attributes:
    - r_regionkey INTEGER (primary key)
    - r_name VARCHAR
    - r_comment VARCHAR

- Table *Nation* with the attributes:
    - n_nationkey INTEGER (primary key)
    - n_name VARCHAR
    - n_comment VARCHAR
    - n_regionkey INTEGER (foreign key)

The reference n_regionkey represents the relationship between the Nation and Region tables including the cardinalities 1..1 for the table Region and 1..MANY for the table Nation.

In the sequel it will be shown how this knowledge is represented inside the OWL ontology.

Source tables:

The data source tables are represented as ontology classes (owl:Class).

```
<!--definition of the ontology class that represents source table Region-->
<owl:Class rdf:ID="Region">
…
 </owl:Class>
```

```
<!--definition of the ontology class that represents source table Nation-->
<owl:Class rdf:ID="Nation">
 …
</owl:Class>
```

Attributes of the source tables:

The attributes of the source tables are represented as different datatype properties inside the ontology. In the definition of the datatype property the *domain* class is the ontology class that represents the source table of the given attribute, while the *range* represents the datatype of the attribute.

On the examples below it can be noticed that the names of the ontology datatype properties differ from the actual source attribute names. This is the vocabulary of the domain that is agreed to be used for constructing the ontology and later for providing the business requirements. Actually, the datatype property names in the ontology are built from the source attribute names by adding the name of the table as a prefix and the keyword ATRIBUT as a suffix.

```
<!--Definition of the attribute r_regionkey, i.e., primary key, of the table Region -->
<owl:DatatypeProperty rdf:about="#Region_r_regionkeyATRIBUT">
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
   <rdfs:domain rdf:resource="#Region"/>
</owl:DatatypeProperty>
```

```
<!--Definition of the attribute r_name of the table Region -->
<owl:DatatypeProperty rdf:about="#Region_r_nameATRIBUT">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Region"/>
</owl:DatatypeProperty>
```

```
<!--Definition of the attribute r_comment of the table Region -->
<owl:DatatypeProperty rdf:about="#Region_r_commentATRIBUT">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Region"/>
</owl:DatatypeProperty>
```

```
<!--Definition of the attribute n_nationkey, i.e., primary key, of the table Nation -->
<owl:DatatypeProperty rdf:about="#Nation_n_nationkeyATRIBUT">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#Nation"/>
</owl:DatatypeProperty>
```

```
<!--Definition of the attribute n_name of the table Nation -->
<owl:DatatypeProperty rdf:about="#Nation_n_nameATRIBUT">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Nation"/>
</owl:DatatypeProperty>
```

```
<!--Definition of the attribute n_comment of the table Nation -->
<owl:DatatypeProperty rdf:about="#Nation_n_commentATRIBUT">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Nation"/>
</owl:DatatypeProperty>
```

```
<!--Definition of the attribute n_regionkey of the table Nation that is actually the foreign key referencing to the table Region -->
<owl:DatatypeProperty rdf:about="#Nation_n_regionkeyATRIBUT">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#Nation"/>
</owl:DatatypeProperty>
```

Source associations:

Since the source table Nation includes the reference to the source table Region (foreign key), this relationship should be explicitly represented inside the ontology. For explicit representation of the relationships between the source tables, the Object Property is introduced in the ontology. Inside the definition of this Object Property the domain class is the one that represents the referencing source table and the range class is the one that represents the referenced source table.

The name of the property is arbitrary but due to common understandability it is advised to be meaningful and comprehensive.

```
<!-- Definition of the IsFrom relationship between the tables Nation and Region -->
<owl:ObjectProperty rdf:about="#IsFrom">
  <rdfs:domain rdf:resource="#Nation"/>
  <rdfs:range rdf:resource="#Region"/>
</owl:ObjectProperty>
```

Cardinalities of the source tables:

For the relationship between tables Region and Nation, the cardinalities are defined to be 1..1 and 1..* respectively. The cardinalities of the source tables inside the defined relationships are represented as subclasses, more precisely restrictions, inside the class that represents those source tables. Therefore, the definitions of the ontology classes that represent source tables are extended with the corresponding restrictions.

Since the definition of the cardinality restriction in the OWL ontology requires the exact integer value, for the purpose of representing the maximum cardinality of MANY the integer value '-1' is used.

```
<!-- complete definition of the concept that represent source table  Region -->
<owl:Class rdf:ID="Region"> …
<!--definition of the maximum cardinality  restriction on the property  IsFrom -->
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty>
     <owl:ObjectProperty rdf:ID="IsFrom"/>
    </owl:onProperty>
    <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:maxCardinality>
   </owl:Restriction>
  </rdfs:subClassOf>
…
<!--definition of the minimum cardinality  restriction on the property  IsFrom -->
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:minCardinality>
    <owl:onProperty>
     <owl:ObjectProperty rdf:about="#IsFrom"/>
    </owl:onProperty>
   </owl:Restriction>
  </rdfs:subClassOf>
…
 </owl:Class>
```

```
<!-- complete definition of the concept that represent source table  Nation -->
<owl:Class rdf:ID="Nation"> …
<!--definition of the maximum cardinality  restriction on the property  IsFrom -->
<rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty>
     <owl:ObjectProperty rdf:ID="IsFrom"/>
```

```
      </owl:onProperty>
      <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >-1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
 <!--definition of the minimum cardinality  restriction on the property  IsFrom -->
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:minCardinality>
    <owl:onProperty>
     <owl:ObjectProperty rdf:about="#IsFrom"/>
    </owl:onProperty>
   </owl:Restriction>
  </rdfs:subClassOf>
…
 </owl:Class>
```

3.1.1.2. Source Mappings

This part of the input represents the structured way of defining mappings which relate ontology representation of data sources, to the available, real data stores. Source mappings are defined inside another XML structure. The graphical representation of the DTD of this XML is provided in figure 8, while the original DTD is provided in Appendix B. Each concept from the ontology may have its mapping inside of this structure. In order to distinguish kinds of the attributes in the sources, different kinds of mappings exist.

All the examples presented below are from the TPC-H relational schema and the ontology produced to represent this schema, with the requirements used in the demo, presented in Appendix A

- The ***mapping of an ontology class*** to the data source table is structured in the following way.

```
<OntologyMapping sourceKind="relational">
    <Ontology type="concept">
        http://www.owl-ontologies.com/unnamed.owl#Nation
    </Ontology>
    <RefOntology type="property">
        http://www.owl-
ontologies.com/unnamed.owl#Nation_n_nationkeyATRIBUT
    </RefOntology>
    <Mapping>
        <Tablename>nation</Tablename>
        <Projections>
```

```
                    <Attribute>n_nationkey</Attribute>
                </Projections>
            </Mapping>
        </OntologyMapping>
```

While the ontology class that is mapped to the source table is identified by the content of the Ontology tag, the ontology concept (datatype property) representing this primary key is identified by the content of the RefOntology tag. It should be also noticed here that this mapping includes projection of the attribute that represents the primary key of the source table (n_nationkey).

- The ***mapping of an ontology datatype property*** to the attribute of a source table, is structured in the following way:

```
    <OntologyMapping sourceKind="relational">
        <Ontology type="property">
            http://www.owl-ontologies.com/unnamed.owl#Region_r_nameATRIBUT
        </Ontology>
        <Mapping>
            <Tablename>region</Tablename>
            <Projections>
                <Attribute>r_regionkey </Attribute>
                <Attribute>r_name </Attribute>
            </Projections>
        </Mapping>
    </OntologyMapping>
```

This structure contains the ontology datatype property that this mapping is prepared for, i.e., Ontology tag. Afterwards, it includes the mapping to the data source table. This Mapping includes name of the data source table, i.e., Tablename, and the attribute in the data source table that stores information represented by the ontology datatype property (r_name), i.e., Projections and Attribute tags. This mapping, since it represents property, also contains attribute that represents primary key of the table that this attribute belongs too.

- The ***mapping of an ontology object property*** to the corresponding datasource relationship, is structured in the following way

```
    <OntologyMapping sourceKind="relational">
        <Ontology type="property">
            http://www.owl-ontologies.com/unnamed.owl#IsFrom
        </Ontology>
        <Mapping>
```

```
                <Tablename>nation</Tablename>
                <Projections>
                        <Attribute>n_nationkey</Attribute>
                        <Attribute>n_regionkey</Attribute>
                </Projections>
        </Mapping>
</OntologyMapping>
```

It can be noticed that in the mappings of the association there is the part that defines ontology object property that represents the association, i.e., Ontology tag. Afterwards, depending on the relationship, the table that uniquely defines the association instances is defined with its name, i.e., Tablename and then also the attributes of that table needed in order to query this association. It is possible to express all association mappings in this way, because the associations many-to-many are not considered here, since they violate the constraints of multidimensional design. Within the Projection tags, the attributes that that represent the primary key of the referencing source table (n_nationkey) and the foreign key to the referenced table (n_regionkey) are included.

- The ***mapping of an ontology datatype property*** that represents ***foreign key***, i.e., reference to another concept, to the, attribute of the source table, is structured in the following way:

```
<OntologyMapping sourceKind="relational">
        <Ontology type="property">
                http://www.owl-ontologies.com/unnamed.owl#Nation_n_regionkeyATRIBUT
        </Ontology>
        <RefOntology type="concept">
                http://www.owl-ontologies.com/unnamed.owl#Region
        </RefOntology>
        <Mapping>
                <Tablename>nation</Tablename>
                <Projections>
                        <Attribute>n_regionkey</Attribute>
                </Projections>
        </Mapping>
</OntologyMapping>
```

It can be noticed that this kind of mapping contains the RefOntology tag that, in fact, represents the ontology concept, which represents the source table referenced by the defined foreign key attribute.

All previous mapping structures can contain, selection information, i.e., the discriminant function, which can include attribute, operator and the value according to which the more specific characteristics of a given attribute is defined.

```
<Selections>
    <Selection>
        <Column>l_shipdate</Column>
        <Operator>&gt;</Operator>
        <Constant>'1998-12-01'</Constant>
    </Selection>
</Selections>
```

Considering the case, discussed in more details in section 3.3., when the concept from the ontology cannot be mapped directly to the data sources, the operators for deriving the new mappings from the existing ones should be introduced. These operators are actually those that are, during the stage of *Operation Identification,* considered for generation of the conceptual model of the ETL process.

```
<OntologyMapping sourceKind="relational">
    <Ontology type="concept">
        http://www.owl-ontologies.com/unnamed.owl#Customer
    </Ontology>
    <RefOntology type="property">
        http://www.owl-ontologies.com/unnamed.owl#Customer_c_custkeyATRIBUT
    </RefOntology>
    <Mapping>
        <Mapping>
            <Tablename>legal_entity</Tablename>
            <Projections>
                <Attribute>le_custkey</Attribute>
            </Projections>
        </Mapping>
        <SQLOperator>UNION</SQLOperator>
        <Mapping>
            <Tablename>individual</Tablename>
            <Projections>
            <Attribute>i_custkey</Attribute>
            </Projections>
        </Mapping>
    </Mapping>
</OntologyMapping>
```

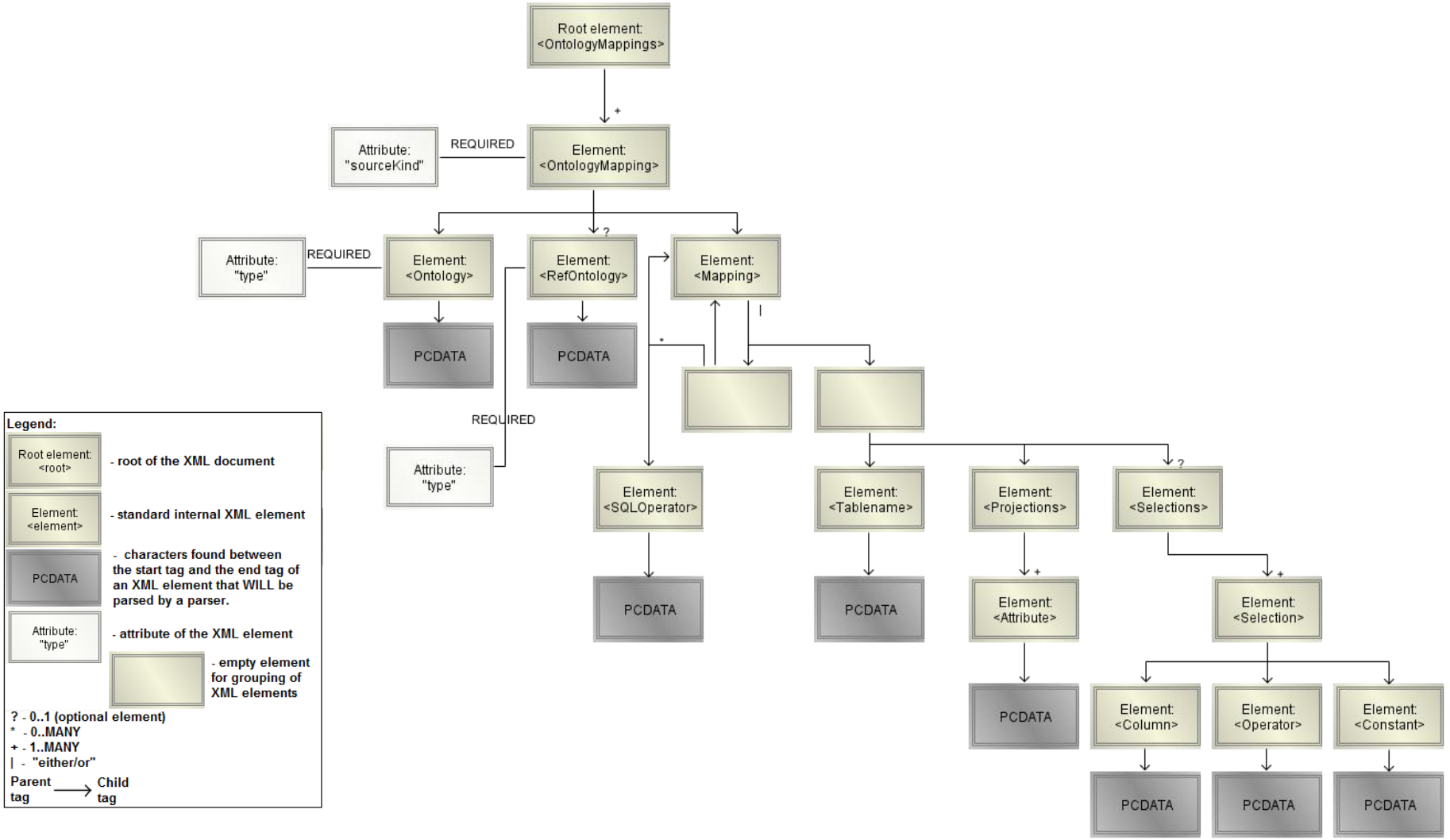


Figure 8: Tree representation of the XML grammar for the source mappings

This example, additionally created concerning the TPC-H schema, represents derived mapping for the ontology concept Customer. For this case, in the ontology, class Customer is declared to have two subclasses LegalEntity and Individual. If the mapping of the concept Customer is not available at the input, new mapping should be derived. If the mappings for the concepts, which are declared as subclasses of the Customer concept, are available, then the new mapping should be derived from these mappings and the additional operator between them, in this case UNION. More details about the rest of the operators are provided in the section 3.3.

3.1.1.3. Business requirements

Design of decision-making systems most frequently starts from a set of specific business needs expressed as requirements. In the case of decision-making systems and underlying data warehouses, the requirements that actually drive their design are the *information requirements*, i.e., the information that the system under consideration should be able to provide. These requirements come in various forms, service level agreements (SLAs), business queries and as already mentioned can be expressed in either structured or unstructured forms. There are many researches handling the problem of capturing the requirements. At this point, it should be noted that the capturing of this kind of requirements (information requirements) is easier than traditional functional requirements gathering, since the people involved in this process are usually business analysis experts who precisely know what kind of information the company really needs. However, the requirement gathering is not the topic of my work and it will be considered that this process has been previously done.

After the business requirements are collected, it is noticed that the identification of multidimensional concepts inside of these requirements is very easy.
For example:
If we have the following business requirement, gathered from a retail company:

- Information about the **revenue** for the company's **shops** in **Barcelona** is required.

The identification of the multidimensional concepts is quite straightforward.

- ***Revenue*** can be identified as a *measure* i.e., the data that is analyzed,
- ***Shops*** can be identified as the *level*, i.e., the perspective from which the data is analyzed, and
- ***Barcelona*** can be identified as the *descriptor* of the level ***city***

Considering the above, the generation of the input XML structures that represent business requirements, according to the previously gathered requirements, is quite effortless.

On figure 9, the graphical representation of DTD, for the XML with business requirements, is provided. The original DTD for the XML with the business requirements is provided in Appendix B.

Since the business specialists involved in the decision-making process, use business terms in their everyday communication, the input business requirements are also expected to be expressed using these terms. On the other side, the sources, created by the DB specialist may not follow the same naming patterns. This gap between the business and the IT world must be bridged, in order for the system to be able to identify real business needs inside the available data sources (ontology). This should be accomplished by introducing the domain vocabulary. This vocabulary should be agreed between these two sides (IT and business) and later used for defining the elements of the corresponding ontology.

Additionally, the structure of these XML files is explained below in more details.

Examples below are captured from the input requirements, used in the demo that is based on the TPC-H schema and presented in the Appendix A.

The XML structure contains four different parts, separated into distinct XML tags.

1. **Levels -** which represent the perspectives used to analyze the data. These are indeed used to view data at a specific level of detail.

```
<dimensions>
    <concept id="Nation_n_name" />
</dimensions>
```

2. **Measures** - which represent summary data that is analyzed. This concept in the requirement structure includes the function based on which the value of the measure is calculated (*<function>).*

```
<measures>
    <concept id="revenue">
        <function>
            Lineitem_l_extendedpriceATRIBUT*Lineitem_l_discountATRIBUT
        </function>
    </concept>
</measures>
```

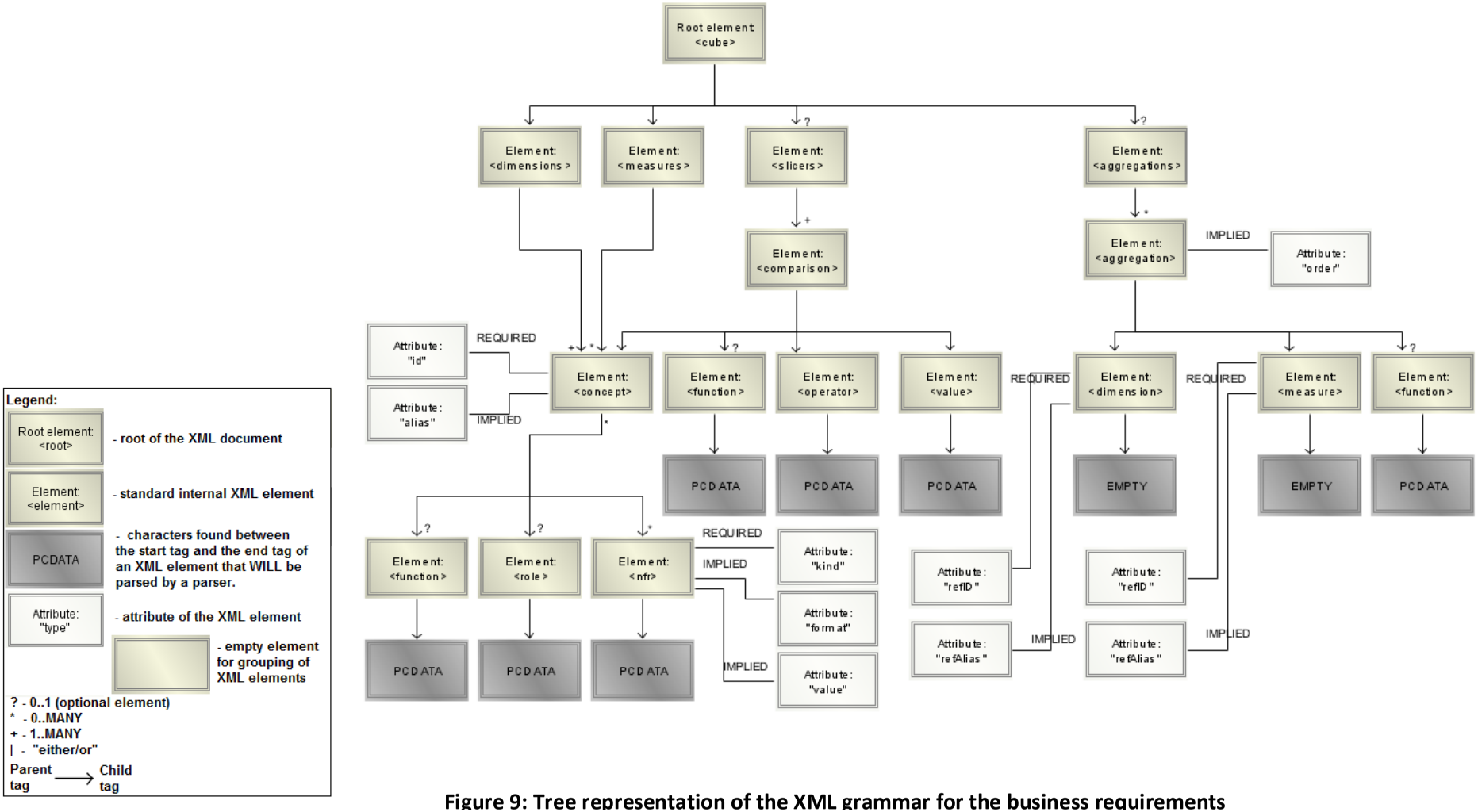


Figure 9: Tree representation of the XML grammar for the business requirements

3. **Levels -** which represent the perspectives used to analyze the data. These are indeed used to view data at a specific level of detail.

```
<dimensions>
    <concept id="Nation_n_name" />
</dimensions>
```

4. **Measures** - which represent summary data that is analyzed. This concept in the requirement structure includes the function based on which the value of the measure is calculated (*<function>).*

```
<measures>
    <concept id="revenue">
        <function>
            Lineitem_l_extendedpriceATRIBUT*Lineitem_l_discountATRIBUT
        </function>
    </concept>
</measures>
```

5. **Descriptors** or **Slicers** - which carry out selections over level data. These concepts limit the resulting set based on the comparison operator. Additional information inside the slicers is the function that can be applied over the concept (extracty_year). It should be noted that this function is different from the function stated in the part related to measures. Main difference is that previous function is used to calculate the value of the concept and thus is placed between the concepts tags, while latter one is actually applied over the concept in order to extract particular information from it. Comparison operators (>, <, ≥, ≤) are due to XML limitations expressed with escape characters (>, <, >= and <=).

```
<slicers>
    <comparison>
        <concept id=" Orders_o_orderdateATRIBUT " />
        <function>extracty_year</function>
        <operator>&lt;=</operator>
        <value>1998</value>
    </comparison>
</slicers>
```

6. **Aggregations** which contain aggregation functions that should be additionally applied over the measures grouped by all the dimensions. Among the aggregation functions the partial order is also defined. This order is needed for

the case when we perform different aggregations. (e.g. for system to be able to distinguish between “average of sums” and “sum of averages”). For each measure and dimension there is one aggregation.

```
<aggregations>
    <aggregation order="1">
        <dimension ref=" Nation_n_name "/>
        <measure ref="revenue " />
        <function>SUM</function>
    </aggregation>
</aggregations>
```

As it can be seen on figure 9, the element *concept* may contain optional attribute *alias* and along with it the optional internal element *role.* This optional information is additionally introduced because it is noticed that the user may ask for the same type of information related to different concepts.

```
<dimensions>
    <concept id="Nation_n_nameATRIBUT" alias="n1">
        <role>Customer</role>
    </concept>
    <concept id="Nation_n_nameATRIBUT" alias="n2">
        <role>Supplier</role>
    </concept>
</dimensions>
```

In this example, it can be seen that there may exist two requirements for the name of the nation. However, it can be also noticed that the first requirement considers the nations of the company’s customers (*Customer*) and the second one considers the nations of the company’s suppliers (*Customer*).

Each concept, besides information requirements, which are captured by identifying the measures and dimensions of interest, can also include non-functional requirements. Non-functional requirements are expressed as separated tags inside the concept tag it refers to.

```
<measures>
    <concept id="revenue">
        <function>
            Lineitem_l_extendedpriceATRIBUT*Lineitem_l_discountATRIBUT
        </function>
        <nfr kind="freshness" format="HH24:MM:SS">&lt;00:10:00</nfr>
    </concept>
</measures>
```

This example shows that a *measure* concept can include non-functional requirement, in this case "*freshness*". Definition of the non-functional requirements includes attributes kind and format in which the value is expressed, and the value of the requirement is between the tags.

Depending on the kind of requirement, non-functional requirements can affect different levels of design. From the last example, the freshness requirement that indicates how often an ETL process should run in order to keep the data warehouse updated, actually affects the execution level and hence it should be considered at the physical level of design. This indeed means that this information should be forwarded to the physical level, i.e., the level on which it should be considered. Another advantage of this principle is that the designers of the further levels (logical or physical) do not need to spend additional time on interpreting the original set of business requirements.

3.1.2. Stages inside the GEM framework

Since the previous sections cover the structures that can be expected as an input of the GEM framework, the following sections cover the stages through which the system has to pass in order to translate requirements from the inputs into the appropriate MD and ETL designs.

In order to accomplish generality of the framework input formats, input concepts, extracted from business requirements, have to be primarily identified in the given sources represented by ontology, tagged with their potential multidimensional roles and appropriately mapped to the data sources. This is done in the ***Requirement Validation*** stage. Afterwards, in the ***Requirement Completion*** stage, the system identifies how these concepts are related in the sources. These two stages represent the main contribution of my work, towards the generalization of the input formats, since the outputs of these stages are the structures that are the inputs for the stages responsible for final generation of both MD and ETL designs (***Multidimensional Validation*** and ***Operation Identification***).

3.1.2.1. Requirement Validation

Requirement Validation represents the initial stage of the GEM framework. Considering the inputs discussed in section 3.1.1., during this stage, the system validates the input business requirements with regard to the available data sources. The system reads the input XML file with business requirements (section 3.1.1.3) and extracts the individual concepts from the business requirements. The ontology used for representing the data stores (section 3.1.1.1) is then searched for these concepts. The identified ontology concepts are then tagged with three kinds of labels: Levels, Measures, or Descriptors. This tagging actually depends on the part of the input XML structure that the concepts

belong to, and it represents the multidimensional role that those concepts have. Therefore, at this place only the concepts that are required at the inputs are tagged with their multidimensional role.

In the sequel, the system reads the XML structure with the source mappings (section 3.1.1.2.) and tries to identify the mapping for each ontology concept found within the business requirements.

Considering the mentioned possibility of the mismatch between the vocabulary used for making the business requirements and the vocabulary for representing data stores, another effort inside the GEM system has been done in overcoming this issue. This effort is actually done through the source mapping process, where the system searches particular relations of the concept (subsumption, synonyms) in order to find the most suitable mappings for this concept.

Therefore, the mapping of the identified concepts to the sources can be either:

- *Direct*, i.e., available in the input source mapping structure, or
- *Derived* by means of ETL operators.

If the mapping for some concept is not directly available inside the input XML structure, then the system should search for alternative solutions.
These alternative solutions, i.e., *derived mappings*, can be deduced from:

- the mapped *subclasses*,
- the mapped *superclasses*, or
- the *possible synonyms* of the given concepts.

Original solution that was proposed, considered that the system was supposed to solely find mapped subclass/supperclass concepts and derive new mappings automatically. However, during my study, I noticed that these automatically derived mappings may not actually fulfill the needs of the designer and initial business requirements.

Therefore, new alternative solution has been proposed and it considers that the system is first supposed to find mapped subclass/supperclass concepts, but then it only needs to propose these concepts to the user along with the possible operations needed for derivation of the new mapping (UNION, INTERSECTION, MINUS, SELECTION, RENAMING).

The advantage of this approach is that the ambiguity in the mapping derivation process is overcome with the manually derived mapping that the user is supposed to provide following the system's suggestions. This ambiguity is especially present when the mapping is derived from the superclasses of the given concept, because in that case, as

we will see later, the three different possible solutions (operations) are available. However, the apparent downside of it is that the automation of the complete GEM process is decreased with this task that is manually executed.

For each individual concept extracted from the business requirements and tagged with the appropriate label the process of the mapping to the sources is triggered.

The complete algorithm is presented in the sequel:

1. if the tagged concept is mapped to the sources then no further action is needed
2. else if the tagged concept is involved in a concept taxonomy then
   (a) if any of its subclass(es) has (have) a mapping then the system proposes derivation of the new mapping considering the mapped subclasses and operation 'union'

   (b) else if any superclass has a mapping then

      i. if the tagged concept has one superclasses and if discriminant function has not been specified in the input XML file then user feedback is required

      ii. if the tagged concept has several superclasses then the system proposes derivation of the new mapping considering the mapped superclasses and operations 'minus' or 'intersection'.

3. else if exists a (transitive) one-to-one association to a mapped concept then suggest it as a potential synonym
   (a) if the suggestion is accepted then the functional requirement is updated with the synonym concept

4. else the concept is not available in the data sources

During steps 2(a) and 2(b), user feedback is required in order to provide mapping for the concepts that are not directly mapped. The system first identifies, in 2(a), which of the concept's subclasses are directly mapped. Then it asks the user to provide new mapping but according to its suggestions. These suggestions contain mapped ontology subclass concepts and operations that the user needs in order to derive new mapping. The same holds for 2(b), with the difference that the system here tries to identify superclasses of a given concept that are directly mapped. After the user loads new mappings, the system continues with the validation process. If there are not any sub/super class that is mapped then the system tries to infer the possible synonyms of the given concept. Possible synonyms of a concept are those concepts that are related to a given concept by means of one-to-one association. The process of identifying synonyms uses the transitivity rule of the ontology to search for the possible synonyms. After the list of

possible synonyms is prepared, system suggests to the user to choose one of them. After the choice has been made, the system continues the process of validation.

As discussed, both through the state of the art and in section 4.2.5 dedicated to the OWL, one of the main reasons for choosing OWL as the ontology language is because of existing support. This support is reflected by the fact that the most of the reasoners that exists support the inference process inside the OWL ontologies. At this point, it can be noticed how we benefit from this support. Relations between the concepts in the ontology (subsumtion and transitivity relationships) that are supposed to be identified in the process of source mapping are actually inferred with the usage of the supporting reasoner.

After all the input concepts are validated and all the mappings are identified, the system produces the structure that represents the subset of ETL operations that are extracted from the mapping information. This structure is part of the input for the latter Operation Identification stage. For concepts that are directly mapped, only an EXTRACTION operation is identified. For those concepts whose mappings are derived from its sub/super classes, the more complex structures are required. One EXTRACTION operation is required for each concept that represents mapped sub/super class in the derived mapping. Then for each additional operator (UNION, INTERSECTION, MINUS), among the mapped concepts, the new corresponding operation is required. Additionally, a SELECTION operation is required if the mapping contains a discriminant function. This structure, at the end, represents the set of the ETL operations that are needed to extract the required concepts from the data sources and it will be included in the output ETL process design.

3.1.2.2. Requirement Completion

After the concepts from the business requirements are identified in the ontology and tagged with the corresponding labels, system starts the second stage, i.e., the Requirement Completion stage. Main task of this stage is to relate already tagged concepts, regarding the sources. Therefore, during this stage, using the ontology, the system completes the set of initial requirements regarding the available sources. Intermediate concepts, which are not initially identified as business requirements, are now identified, since they are needed to retrieve actually required information. The system, in the process that follows the requirement completion process, i.e., *Annotating the ontology AOS*, tries to tag all intermediate concepts with their possible labels. However, the tagging of all concepts is finalized during the Multidimensional Validation stage. The main output of the Requirement Completion stage is the graph structure that contains the paths identified between the tagged concepts.

Pruning process

Completion stage starts with the pruning process, disregarding the concepts or associations that do not fulfill certain rules.

(a) System disregards concepts or relationships that are neither mapped nor tagged. However, if concept taxonomy is affected, the concept is replaced with the first superclass that is mapped or tagged.

(b) System prunes all the mapped many-to-many associations (i.e., *-*). Such associations violate the three summarization necessary conditions, and therefore, they cannot be considered for the resulting multidimensional design. These conditions are later discussed in the section 3.1.2.3., dedicated to Multidimensional Validation stage.

Output of the pruning process, is a subset of the input ontology, called AOS (**A**nnotated **O**ntology **S**ubset).

Looking for paths between tagged concepts

Since it is possible to represent an arbitrary ontology as a graph, then the ontology concepts are considered as nodes and ontology associations are considered as edges. According to this formulation, inside the AOS, the system will try to identify paths between the nodes associated to the tagged concepts.

For identifying how tagged concepts are related in the sources, the system uses the following algorithm that computes paths between those concepts.

```
1. foreach edge e in O do
    (a) if right_left_concepts(e) are tagged then paths_between_tagged_concepts ∪=e;
    (b) else if right_concept(e) is tagged then max_length_paths ∪=e; //Seed edges
2. while size(max_length_paths) != Ø do
    (a) paths := Ø;
    (b) foreach path p in max_length_paths do
            i. extended_paths := explore_new_edges(p, O); //only considering edges not in p
            ii. foreach path p1 in extended_paths do
                    A.if left_concept(p1) is tagged then
                            paths_between_tagged_concepts ∪=p1;
                    B.else paths ∪= p1;
    (c) max_length_paths := paths;
3. return paths_between_tagged_concepts;
```

System, in the step 1(a), starts by identifying edges that relate tagged concepts directly and, in the step 1(b), edges reaching tagged concepts (seed edges). Although, the edges in the AOS are not directed, it will be considered that the node, representing the tagged concept, is in the right-end of the seed edge, and that its counterpart is in the left-end.

Afterwards, this algorithm applies the transitivity rule starting form the tagged concepts. At the first iteration, the system, in the step 2(b)i, explores new edges such that their right-end matches the left-end of a seed edge, and similarly for the forthcoming iterations. Incrementally, system explores paths, starting from the tagged concept, adding a new edge per iteration. There are two main restrictions during this path exploration process that need to be considered:

1. in a given path, already explored edges cannot be explored again
2. if system reaches another node, representing a tagged concept, in the step 2(b)iiA, it finishes exploring that path (i.e., the path between those tagged concepts is completely identified)

It can be noticed, from the steps 1(b) and 2(c), that in the given iteration i, the system only explores the longest paths computed in the previous iteration. Eventually, all paths are explored and the algorithm finishes.

It should be stated that this algorithm is *sound* since it computes direct relationships and propagates them according to the transitivity rule, and *complete*, because it converges. Additionally, it can be noticed that each path is explored only once. This algorithm has theoretical exponential upper bound regarding the size of the longest path between tagged concepts. However, this theoretical upper bound is hardly achievable in real-world ontologies, since they have neither all classes with maximum connectivity nor all paths are of the maximum length, especially without all the many-to-many relationships, previously pruned.

Producing the Output Subset

Taking the set of the paths between tagged concepts, created in the previous phase, the following algorithm forms a subset of ontology concepts that are actually needed to answer functional requirements. This subset consists of those concepts that are initially tagged and of those concepts (intermediate concepts) that exist in the paths between the tagged ones. This algorithm might require user feedback, for making precise decision, about the ontology subset that is going to represent the final output. If between two tagged concepts there are more than one path then we ask the user for disambiguation (i.e., which is the path fulfilling the semantics needed to answer real business needs).

The final AOS produced in this phase is compound by the paths selected by the user.

Annotating the ontology AOS

After the system provides the complete AOS, including new concepts, needed to answer the functional requirement, it must be checked that the whole graph makes MD sense.
First, the system checks the semantics of each edge according to the tags of the related concepts, if those tags are defined, and the edge multiplicity. According to this, the system tags each edge with multidimensional relationship it may represent. Afterwards, the system considers the nodes in the graph, i.e., *factual nodes* – nodes representing concepts tagged as measures and *dimensional nodes* – nodes representing concepts tagged either as levels or descriptors. Relations between these nodes need to be checked in order for system, to guarantee that the MD design principles have been obeyed. For example, factual data cannot be related to dimensional data by means of a one-to-many association, as by definition, each instance of factual data is identified by point in each of its analysis dimensions. If the dimensional data appear on the *-end of the relation, the other end of the relation has to be also tagged as dimensional data. Furthermore, associations accepting zeros cannot appear in the dimensional end either, as they do not preserve *completeness*.

The system, therefore, analyzes the graph and if it finds edges that violate any of the mentioned conditions, it tries to fix that incorrectness. For example, if the node is in the many-end of a many-to-one association and is tagged as dimensional, then its counterpart should also be dimensional. If by doing so, system is able to infer an unequivocal label, this knowledge is propagated to the rest of the AOS. However, if it is not possible to identify correct combination of tags among the related concepts, the algorithm stops and system suppose to offer alternative scenarios. Appropriate techniques for this task are described in [7].

At this point, after the previous two stages are finished the system provides the structures that are actually needed for the next two stages of the GEM framework that are aimed at finalizing the designs of the multidimensional schema and supporting ETL process.

First, as a result of the Requirement Validation stage the structure containing the set of the ETL operations, needed for extracting data from the available sources, is created. This structure is passed to the stage of Operation Identification. Another structure produced in the Requirement Completion stage is the graph structure containing the paths between the tagged concepts. This structure, after all the concepts have been appropriately tagged, indeed represents potential MD design. This design, after validated

in the stage of Multidimensional Validation, represents the actual resulting MD design needed for answering the input business requirements. On the other side, along with the ETL structure, this MD design represents the input of the Operation Identification stage. Operation Identification stage after considering operations for extracting the data from the sources and generating new ones according to the relations between those data (found in the MD design), as the output generates the design of the ETL process needed for populating the produced MD schema design.

3.1.2.3. Multidimensional Validation

The main task of this stage is the validation of the potential MD design produced in the previous stage. During validation, system checks whether its concepts and associations collectively produce a valid data cube, according to multidimensional design principles.

Two main issues are checked:

i) Whether the factual data is arranged in a MD space (i.e., if each instance of factual data is identified by a point in each of its analysis dimensions), and

ii) Whether the summarization is correct by examining following conditions proposed in [6]:

   (1) *disjointness* – the sets of objects to be arranged must be disjoint

   (2) *completeness* - the union of the subsets must constitute the entire set

   (3) *compatibility* of the dimension with the type of the measure being arranged and the aggregation function

If the initial validation fails, i.e., if the previous constraints are not satisfied, the system should be able to propose alternative solutions. Otherwise, the resulting schema is directly derived from the AOS.

During the final part of the requirement completion stage, system might have propagated some tags, while tagging the AOS associations. However, this does not guarantee that all concepts have a MD label at this point. Therefore, this stage starts with the pre-process aimed to derive new MD knowledge from non-tagged concepts, and each non-tagged concept is considered to play a dimensional or factual role and thus, it can be tagged as dimensional/factual node. Next, the system validates if any of these tags, eventually, are sound in a MD sense. In this step, system determines every potential MD tagging that would make sense for the input functional requirements and also determines how this alternatives would affect output schema. Positive aspect of this is that system might derive some interesting and maybe useful analytical options that may have been overlooked by the designer.

For each possible combination of tagging, provided by the previous step, which does not contradict the edge semantics already existing in the AOS, an alternative annotation is created. The system then starts validation of each AOS, and considers only those that make MD sense. Consequently, with the single functional requirement at the input, the system may produce several valid MD conceptual designs at the output. Afterwards, each resulting MD design causes the new invocation of the Operation Identification process and the generation the different ETL process design.

3.1.2.4. Operation Identification

This stage represents semi-automatic process of ETL operation identification. It comprises three phases. As an input, this stage takes one validated graph, produced in the previous stage and the initial set of the operations produced at the end of the Requirements Validation stage and launches the same identification process.

Phase I

This phase identifies operations that are needed for mapping the concepts to target data stores. For fulfilling this task the system starts from the operations identified in the Requirement Validation stage and it uses the target schema produced in the previous stage.

During this phase, the system mainly identifies schema modification operations as follows.

- *Selection* is generated from the concepts having attached a selection condition, or when a required concept does not have any mapped source (neither it nor its subclasses), while some of its superclasses do have such mapping (step 2(b) of Requirement Validation algorithm, section 3.4.).
- *Union* appears when a required concept is not directly mapped to the sources, but through its mapped subclasses (step 2(a) of Requirement Validation algorithm, section 3.4.).
- *Intersection* and *Minus*, similarly as Union, are generated when the concept is not mapped directly, but through its mapped supperclasses (step 2(b)ii. of Requirement Validation algorithm, section 3.4.).
- *Join* is generated for every association in the ontology. Additionally it can be marked as *outer* if one or both association ends allows zeros.
- *Aggregation* is generated when a many-to-one association is found so that there is a measure at its many-end.
- *Renaming* is generated for each attribute in the data sources and gives to it the name of the corresponding ontological concept.
- *Projection* is generated for each concept and association in the ontology

- *Function* expresses operations stated in the requirements.

Phase II

During this phase, designer has a chance to refine the previously produced design, concerning additional information from data sources that can be useful.

For example, the domain ontology might relate state with zip code and street address. If there is a source containing information about "location" and contains both the street address and zip code in the same field, then such information is definitely useful, but the domain ontology cannot help. We can correct this by enriching the result with producing the appropriate extraction function(s). Nf-reqs can be exploited in a similar way.

Phase III

The last phase of operation identification stage complements the design with operations needed to satisfy standard business and design needs. This phase automatically identifies typical DW operations that can be added to the design in the later stages. The list of these typical operations can go long and it is claimed that the provided method is extensible to adapt this list.

3.2. My contributions to the GEM framework

As my thesis covers the initial stages of the framework, my major contribution to the GEM is the development of Requirement Validation and Requirement Completion stages. Realization of these tasks indeed generalizes the input formats of the GEM framework and is supposed to additionally higher the amount of the overall automation. This is achieved through these two stages in a way that from the inputs, that in a general way represent available data sources (OWL ontology and XML source mappings) and specific business requirements (XML), the system generates the structures that are necessary for producing the output MD and ETL designs. The additional automation is achieved from the fact that the system receives the structured requirements, and using the inference possibilities (reasoner) of the available ontology, automatically deduces the relations between the concepts extracted from the requirements inside the available sources. Furthermore, according to these relations and the information how these concepts can be extracted from the sources (source mappings) the system produces the appropriate MD and ETL designs.

My additional contribution to the GEM framework is complete integration of the parts that are newly implemented by myself, with the parts that have already been implemented by professor Oscar Romero and Daniel Gil Gonzalez from the Technical University of Catalonia.

4. GEM development

Since the previous chapter represents detailed description of new GEM approach and supporting framework, with the main focus on the stages of the framework that I developed, this chapter corresponds to detailed and formal description of the process during which these stages of the framework were developed. Development process represents the second (technological) part of my master thesis.

4.1. Development Method

As a software development process, the process of implementation of the GEM features should follow and respect some basic principles of the Software Engineering. Therefore, prior the beginning of the development process, the possible development method that would appropriately suit the needs of this kind of project, was discussed. Traditional plan-driven software development methods usually require the set of requirements to be fully available prior the begging of the design and implementation phases and for each phase of the development to be strictly respected. Considering the academic and research environment, ad-hoc team organization and low overall criticality of this project, it was concluded that the method that would best suit this project, is Agile Software Development method. In the following sections, the brief introduction to Agile Development, its suitability to this project and more detailed reasons for choosing it, will be provided. Furthermore, since there are various Agile Software Development methods available, the one that best suits this project will be discussed along with its necessary adaptation to the academic and research environment.

4.1.1. Agile Software Development Methods

Agile Software Development considers the set of software development methods that are primarily based on incremental software development. This considers that the requirements and outputs of the implementation phases evolve during the development process. These methods belong to the group of so-called lightweight development methods and they are formally defined in 2001 through the Manifesto for Agile Software Development [15].

Manifesto emphasizes the four main principles that Agile Software Development Method underlies on:

1. Individuals and Interactions – which means that inside the projects that follows Agile Software Development method, individualism and self-organization, are important as much as the easy and effortless communication inside the team.
2. Working software – which means that during these projects at the end of each iteration, the proper working piece of software should be available in order to justify

the work during that iteration and in order to minimize the risk of non-understanding the starting requirements.

3. Customer collaboration – which means that instead of gathering the whole set of the requirements at the beginning, the collaboration with the customer representative should be constant during the whole project and the requirements should be constantly gathered prior the beginning of each iteration.
4. Responding to change – which means that the project team should be feasible to quickly respond to sudden changes during the project

Some characteristics of agile software processes, from the fast delivery point of view are given in [18].

- Agile software process supports modular development.
- Short cycles enable fast verification and prompt corrections.
- Agile software processes are open for necessary adaptations.
- Incremental process minimizes the risks and allows creating of the fully functional application through small lighter steps.
- Agile development supports introduction, modification and possible removal of the requirements in successive iterations.

Prior the choice of using one of the Agile Software Development Method in this project, the results of the surveys that was conducted in the few independent studies, and that examines the results of using the Agile methods, were overviewed. One of them, conducted in 2003 by Shine Technologies [16], considering experience of using Agile methods in various environments, showed results that widely support the using of Agile methods. Another survey represented in [17] also examined overall experience and satisfaction with the agile methods used in business development projects and had very similar results, with the concluding comment that “Agile works in practice…. “.

4.1.2. Agile suitability to GEM development

Besides the previously commented Agile-using experience surveys, additionally, the suitability of Agile methods for this project has been also taken into consideration, before choosing one of them for the development process.

In [18], the detailed survey about the available Agile Software development methods is conducted. Considering the development of GEM features, these methods are examined. The method that is found to be the most appropriate for this project is *Scrum.* Scrum actually dates back to 1986, but it was revisited and updated by Schwaber in 1995 and later by Schwaber and Beedle in 2002. The main idea of Scrum, explained in [18], is that system development should involve several environmental and technical variables (technology,

requirements, time frame etc.). These variables make development process unpredictable, which requires flexibility of the process in order to be able to respond to the changes. The scrum process is depicted in Figure 10 [18].

It can be noticed that the scrum process includes three phases: pre-game, development and post-game.

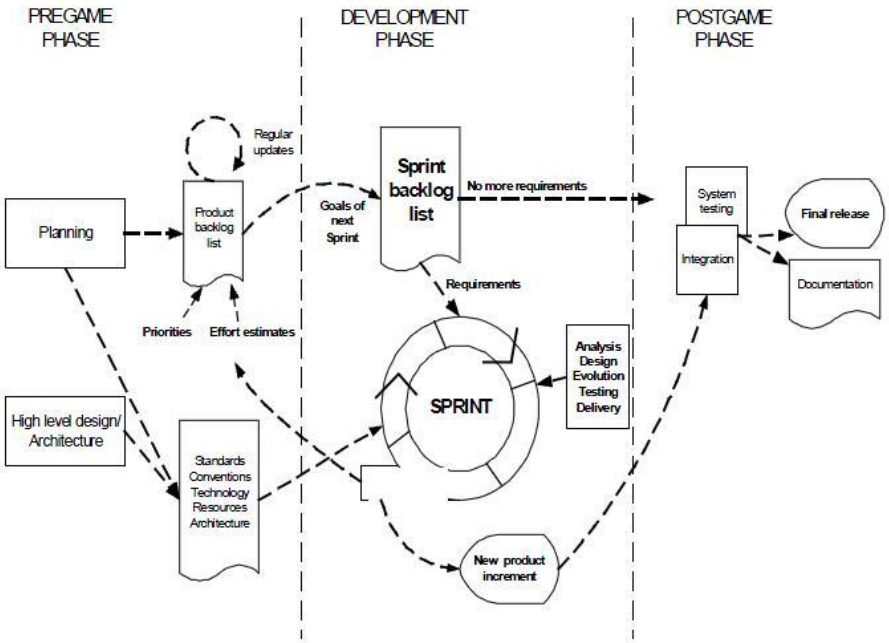


Figure 10: Scrum process, taken from [18]

During the **pre-game phase,** the initial planning and high level designs are provided. This includes complete definition of the system being developed, along with the currently known requirements and concerning that the system architecture is produced. The initial requirements are stored inside the **Product Backlog list**. This set of the requirements and the initial architecture is open to future changes, during the iterations of the **development phase**. Development or **game-phase** actually includes implementation of the system features through the multiple iterations (***sprints***). During these iterations mentioned variables are watched and according to their changes the iterations should be appropriately adapted. The final phase, i.e., **post-game phase**, is one where the complete system functionality is available and it also includes the testing of the developed system.

4.1.3. Reasons for choosing Agile and Scrum

From the general Agile Development principles and Scrum process explained above, it is noticed that the development of GEM features can easily follow this process with the minor adaptation points.

First, the requirement issue was considered. As it is stated, Scrum as the Agile Development Method is suitable for the environments where the set of the requirements is not finite at the beginning (requirements variable). Since GEM project is the part of the research and thus subject to constant changes and improvements, than this perfectly suites its needs. As the part of the master thesis, this project also requires high level of individualism, but also on the other side efficient communication inside the research group. This is another reason for choosing one of the Agile methods, since the first principle of Agile states that the development process should respect both self-organization and cooperation inside the project team. Agile methods also suits the GEM development project, in the terms of the number of team members, since it is commented that Agile methods give better results in the smaller environments. Specifically in [18] it is commented that Scrum is Agile method most suitable for smaller teams.

The only doubt about using the Agile methods is coming from the fact that the members of the teams, as proposed by the Agile Manifest, should be mostly senior developers. Since the project includes students working on their final projects then it can be stated that not all the members of the team are senior high-experienced developers. This issue is compensated by the fact that my work was constantly supervised by the thesis advisors and that the meetings were organized very often in order to keep track of all the obstacles that had arisen. That actually represents another reason for choosing the incremental development process (Agile) since it makes easier for the project supervisors to constantly follow the progress of the development. Furthermore, inside of the development phase of the Scrum process (Figure 10), each iteration (sprint) can include traditional phases of software development: requirements, design, implementation etc. and most importantly this set of phases can be arbitrary adapted to the real project needs.

From the above mentioned, it can be seen that Agile Software development method, more specifically - Scrum, is quite suitable for the development of GEM features.

4.1.4. Scrum adaptation to the GEM development methods

Since this project represents the master thesis project there should be made certain adjustments of the Scrum process to the real needs of this project.

The first adaptation of the Scrum concerns the roles inside the process. The original Scrum includes six different roles inside the process.

- *Scrum Master* is responsible for ensuring that the project is carried through according to the Scrum practices and that it progresses as planned
- *Product Owner* participates in estimating the development efforts and in turning the issues from the Backlog list (requirements) into features to be developed.

- *Scrum Team* is the project team that has the authority to decide on the necessary actions and to organize itself in order to achieve the goals of each iteration (*sprint*)
- *Customer* participates in the task related to product Backlog list for the system being developed.
- *Management* is in charge of final decision making along with the charters, standards and conventions to be followed. It also participates in setting the goals and requirements.

These roles are accordingly arranged and adapted to the members involved in this project. The team of my master supervisors, i.e. professor Alberto Abello and Oscar Romero has been responsible for caring out the project and following its progress, and also participating in the setting of the requirement list. Therefore, the roles that they had in the project correspond to *Scrum Master, Management* and *Customer.* On the other side, my duty inside the development process was first to transform the requirements and theoretical issues into the appropriate features that need to be developed and later to develop those features. Therefore, I was given the role of *Product Owner* and *Scrum Team.*

First, the pre-game phase actually represents the theoretical part of my thesis, since there (in chapter 3) the complete and detailed description of the GEM framework is provided. Along with the description, the low-level design (architecture) of the whole system is provided in Figure 1.

During the development of the GEM framework features that my thesis covers, the several iterations have been identified:

- 1st – Reading and extracting of the inputs
- 2nd – Requirement Validation
- 3rd – Requirement Completion
- 4th – Integration of the MDBE system
- 5th – Integration of the Operation Identification stage
- 6th – Graphical User Interface

All these iterations have been driven by the requirements that are collected both superficially at the beginning of project and incrementally, i.e., more precisely prior each iteration. At the end of each iteration, collective meetings have been organized by the tutors to review the finalized iteration and to discuss arose issues and their possible solutions.

The following sections exhaustively, respecting the development phase of the Scrum process, cover the process of implementation of GEM features that are included in my thesis.

4.2. Generally used technologies

In this section, the general technologies, programming languages and development tools that are used during the development process, will be briefly presented along with the reasons for their selection. Afterwards, during the project iterations the specific technologies will be incrementally introduced.

4.2.1. Visual Paradigm for UML

For designing the modules implemented during the project, the Visual Paradigm for UML is used. This tool is chosen for the design process because it represents a powerful, cross-platform, feature-rich, and most importantly free UML tool (community edition). It supports the latest UML standards. Visual Paradigm for UML Community Edition (VP-UML CE) is chosen and it provides the most easy-to-use and intuitive visual modeling environment. It also provides rich set of export capabilities such as XMI, XML, PDF, JPG and more.

4.2.2. Java

For the implementation of the GEM features, the Java programming language has been chosen. The main reason behind this choice is the fact that other features of the GEM framework has been already implemented in Java, and the fact that Java represents the most common used programming language, which can be justified mostly by its platform-independent and portable nature.

4.2.3. Net Beans 6.9.1 as development environment – IDE

The complete development of the project is realized inside Net Beans 6.9.1. The choice of this development tool can be mainly justified by the fact that I was primarily experienced working with Net Beans IDE, and the fact that none of the IDEs were actually considered as mandatory. Another factor that led to this decision is that Net Beans has better support for drag and drop GUI development.

4.2.4. XML as input format

Being one of the most used forms for exchanging a wide variety of data between systems and on the Web, it is unnecessary explaining XML here. However, at this place it is important to emphasize the advantages that XML offers and that GEM, using it at its input, potentially benefits from. The main goal of XML is to provide the form for communication among different systems, supporting various platforms and as the most important supporting usability over the internet.

As already mentioned, my work inside the GEM project aims at providing GEM with the general input set. This means that the input sources and business requirements must be expressed in a way that can support various forms and technologies. Besides some well known technologies, GEM system is supposed to support various, even unstructured, forms of providing the business requirements (Free-text formats, Web content, email, questionnaires etc.). From this, it can be noticed that XML is the best choice for representing business requirements at the GEM input. On the other side, since no specific technology for the data sources is implied, the mappings of the ontology concepts are also conveniently represented in XML.

4.2.5. OWL Web Ontology Language

OWL is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented. It can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. Inside the GEM project, ontologies are chosen for representing the source data stores. Since the system has to be able to read and infer relations inside the source data stores, the OWL has been chosen, because OWL provides system with the means not only for representing information but also for automatic processing of that information. Another factor for choosing OWL inside GEM, was the fact that was discovered in the state of the art. It is said that the OWL, as the most commonly used language, has the highest support in terms of reasoners. Reasoner, i.e., inference engine, is a piece of software that can be used for automatic inference of the additional knowledge concerning the rules specified by the ontology. Therefore, inside GEM, reasoners will be used for automatic deduction of the knowledge about the given data sources. Also another reason that is mentioned in the state of the art is the fact that OWL is the proposed W3C standard for representing ontologies on the Web.

Since **OWL** tends to be very complex language, it should be stated that only the subset of the OWL elements is actually used for building the ontology that represents data sources in GEM. The elements used inside the ontology are presented in section 3.1.1.1..

4.3. Incremental development of GEM features

As already explained, the first phase of the Scrum process (pre-game phase) is completed inside the theoretical part of the thesis. Therefore, here, the second (development) phase is presented. As, the development process of the GEM features is based on the Scrum, i.e., Agile Software Development Method, its development phase is then divided into incremental sprints or iterations, each one driven by the requirements observed at the beginning of that iteration.
The general structure of one iteration is adapted to GEM needs and it contains the following:

- ***Requirements*** that drive the specific iteration
- Specific ***technologies*** introduced in the iteration
- ***Design*** of the feature(s) covered by the iteration
- ***Implementation***
- ***Obstacles*** that arose during the iteration
- Agreed ***solutions*** for the arose obstacles
- ***Results*** of the iteration

This structure is general and thus, through the iterations it can be modified according to the specifics of that iteration.

Design part of the iteration contains UML diagrams that are used to model the features of the system that are specified by the requirements. Depending on the need and understandability issue some of the following diagrams are provided:

- Use case diagram - for representing requirements
- UML class diagram – for representing structural model of the part developed in the iteration
- UML sequence diagram – for representing the functionality of the part developed in the iteration
- UML activity diagram – for representing the algorithm included in the particular operation

It should be additionally stated that inside the design parts of the following iterations, the external concepts (e.g. from Java libraries and already implemented modules) are depicted with different color (dark gray).

Requirements of each stage were originally discussed on the oral meetings and thus are first given in the non-formal way, but are later translated to the appropriate use case models.

Implementation part covers the work during the implementation process. It also includes translation of the design concepts to the resulting code. Specifics of the Java libraries, needed for the implementation of particular functionalities, are also presented inside the implementation part of the iteration.

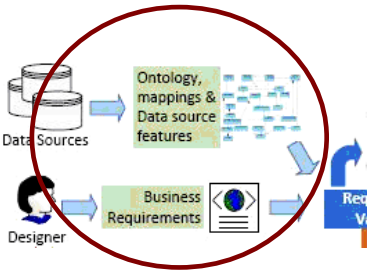
Even though the testing of each iteration has been conducted at its end, complete testing is based on the TPC-H benchmark and therefore, the testing is fully covered in the section that follows the development phase of the GEM. Prior the testing section, the brief introduction to TPC-H benchmark, which is used for the testing purposes, is also presented.

Before the beginning of the development, the full picture of the GEM should be again just briefly considered. Probably the best and the easiest way for making the full picture of GEM system is by reviewing the Figure 1, where the separated stages of the system and communication between those stages can be seen without the need to delve into any formal representation. From this model, the basic set of requirements is extracted, but prior each iteration this set is supplemented with more precise requirements concerning particular iteration.

4.3.1. *1st iteration* - Reading and extracting of the inputs

Requirements

At the beginning of this project, the inputs of the GEM system were considered. Therefore, the following requirements, concerning these inputs, arose in the first iteration of the development process:



- The system should be able to receive two formats of the inputs, i.e., XML and OWL structures.
- The system should be able to read and parse these formats in order to extract useful information.
- The system should be able to extract three different kinds of information. Two structured in XML, i.e., ***business requirements*** and ***source mappings*** and one in OWL, i.e., ontology representing ***source data stores***.
- Content of the input files should be stored into the previously prepared structures and available for later using.

The set of these requirements is expressed with the use case diagram depicted in the Figure 11.

Technologies

Concerning XML input structures, Java API for parsing the XML files is needed. Starting in Java 1.4, Sun bundled the Crimson XML parser and both SAX and DOM APIs into the standard Java class library - **JAXP**. The main difference between SAX and DOM is that in the SAX, only those parts of the document that are actually needed are stored in memory and hence the SAX is quite fast and extremely memory-efficient. SAX is actually based on the events that occur during the reading of the XML file. DOM, on the other side, must keep the entire document in memory at once. Furthermore, the DOM data structures also tend to be less efficient than one of the SAX API. From the above it can be seen that JAXP represents the solid choice for parsing the XML inputs, with its strongest argument of being standard Java class library. Additionally, the choice between DOM and SAX is also quite straightforward concerning efficiency that SAX offers, and thus, the SAX API has been chosen.

On the other side, the means for reading and parsing the ontology represented in OWL was also required. For this purpose **JENA** (Semantic Web Framework for Java) has been chosen. JENA is open source technology and its development has originally established by the Semantic Web Research group at HP Labs. JENA represents a Java framework for building Semantic Web applications. It provides Java programmatic environment for both creation and parsing of various Semantic Web Standards (RDF, RDFS, OWL etc.). Besides its functionality for simple reading and writing of the semantic web documents, JENA also contains features for processing of these documents, i.e., rule-based inference engines (reasoners). JENA framework includes OWL API for handling the ontologies written using OWL Semantic Web Standard. Considering the needs of GEM system, JENA seems to be very suitable for handling input source data stores represented as OWL ontology.

Design

Considering the requirements, the design of the initial part of the GEM system, for reading and parsing input structures, is provided.

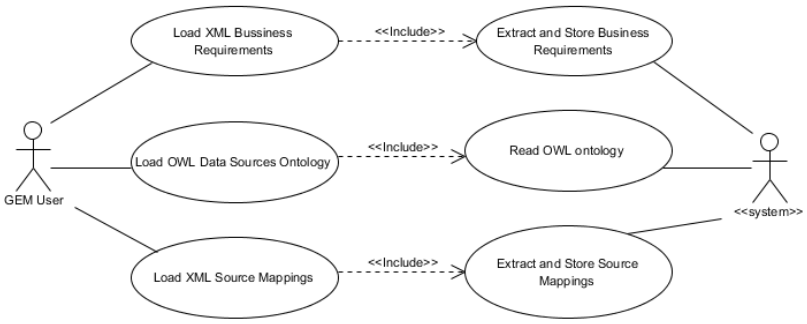


Figure 11: Use case diagram of Input Reading module

Due to the specifics that JAXP and JENA library contains, design of GEM module, for reading the inputs, is slightly driven by these specific needs. Since the information extracted from the input files are supposed to be stored, the structures for storing all these information, should also be designed.

Designing process of the first iteration starts from the requirement that concerns reading and parsing of the XML files that contain **business requirements**. Package that contains complete model of this part (*gem_xml*) is presented in Figure 12. The class modeled for storing all information gathered from the business requirement file, *XMLStruct,* should contain all data of the input XML structure that is discussed in section 3.1.1.3.. Probably the most important is the design of the structures for storing different concepts that can be extracted from the business requirements. Therefore, class *Concept* is introduced in the model, along with its subclasses. Each of these subclasses is intended to represent different multidimensional role that these concepts have, i.e., *Level, Descriptor* (Slicers) or *Measure.*

This taxonomy among the concepts is important because these three types of the concepts can contain different kinds of information, specific for that concept. Descriptor, for example can contain additional function (*functionOverConcept*) that is supposed to be applied over the concept before comparing it with the given value (e.g. extract_year(Date)>2001). *Measure* is also specific because for this kind of concept it should be possible to additionally store the concepts that appear in the function (*functionConcepts*). This function (*function*) is used for calculation of the measure value and thus, it should be differed from *functionOverConcept* which is supposed to be applied over the concept.

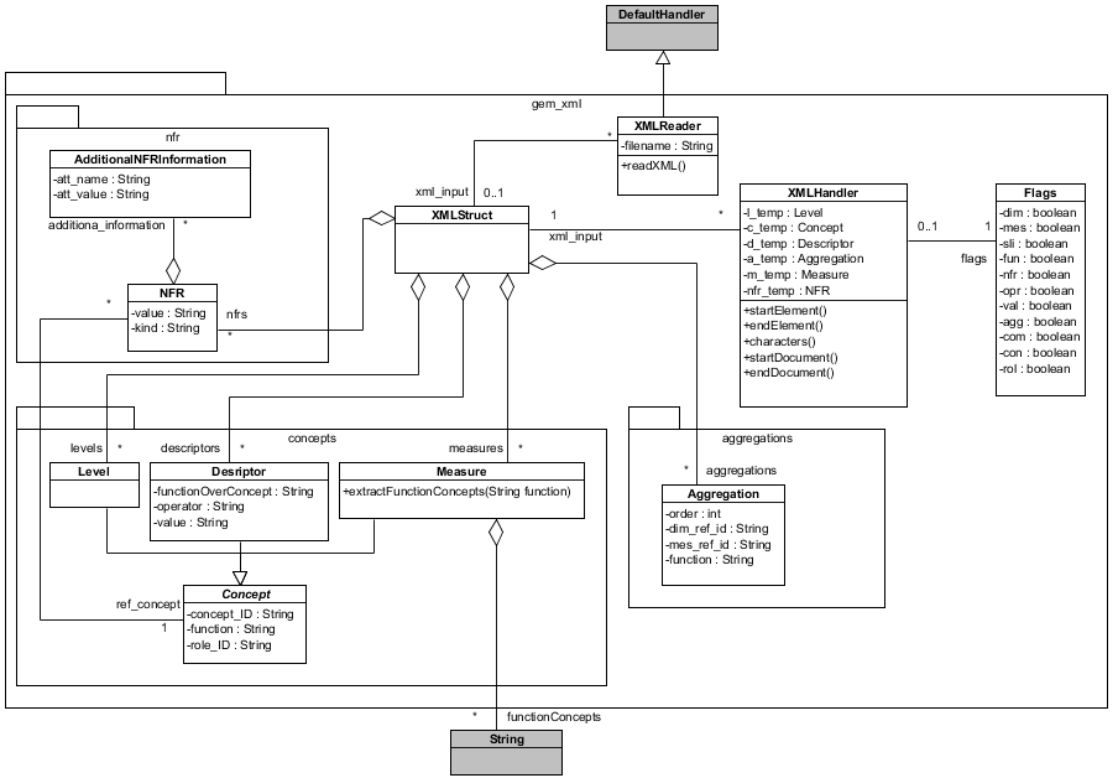


Figure 12: Class diagram for the parsing of XML with business requirements

Classes concerning concepts of the input requirements are included in the package *concepts* that is included in the surrounding *gem_xml* package. Afterwards, the structures for storing non-functional requirements related to the specific concept are provided in the package *nfr*. Aggregation functions for the measure concepts are also modeled with the package *aggregations,* which is along with *nfr* also included in the *gem_xml* package.

After modeling the structures for storing extracted information, the part that actually supports XML parsing should be modeled. Guided by the specification of JAXP, the system should contain the class that extends *DefaultHandler* class form the JAXP library. This class is *XMLHandler. H*andler is needed because SAX API, that is previously chosen, is based on the events that occur during the parsing. Therefore, this handler is used to appropriately react on these events. More details about these events and XMLHandler class are provided in the implementation part of this iteration. The interface to the rest of the system is the class

XMLReader that receives the name of the file containing the XML structure and the process is started with the invocation of its operation *readXML().*

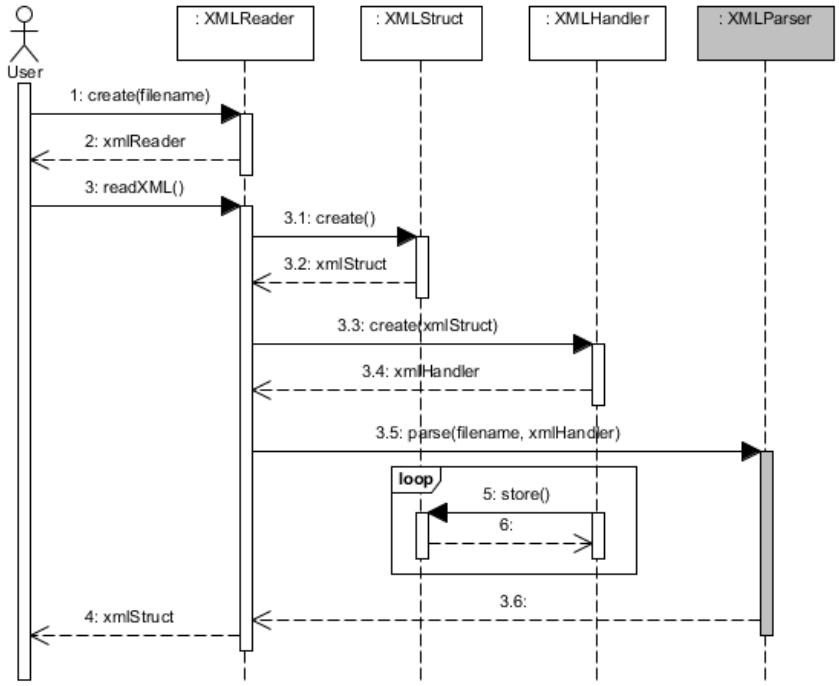


Figure 13: Sequence diagram for Reading XML with business requirements

Complete process of reading input XML with business requirements is depicted in Figure 13 with the corresponding sequence diagram. In the diagram, it can be noticed that the process of reading and extracting of business requirements starts with the creation of *XMLReader* object (*step 1*), as it represents the interface of this part to the rest of the system. This object receives the name of the XML file with the business requirements (*filename*), prepares the structures needed for XML parsing, i.e., the structure for storing the data (*XMLStruct*) and *XMLHandler* for handling the dataflow from XML file to *XMLStruct* object (*step 3.1* and *step 3.3*). *XMLReader* then, using the instance of XMLParser class (from JAXP library), starts the parsing process (*step 3.5*). As it can be seen in the Figure 13, handler object receives the instance of the *XMLStruct*, and thus establishes the communication with that structure. This is required because handler object, during the process of XML parsing, needs to forward data gathered from XML to the corresponding elements of the prepared structure (step 5). At the end of the parsing process, *XMLReader* provides the instance of the *XMLStruct* filled with the data extracted from the XML.

The design of the part, responsible for reading and parsing of the XML structure that contains source mappings, is mostly analogous to the previous one. Difference exists only in terms of the structure designed for storing the extracted data. Since this XML contains information

about source mappings, the design of the corresponding structure is provided and depicted in Figure 14.

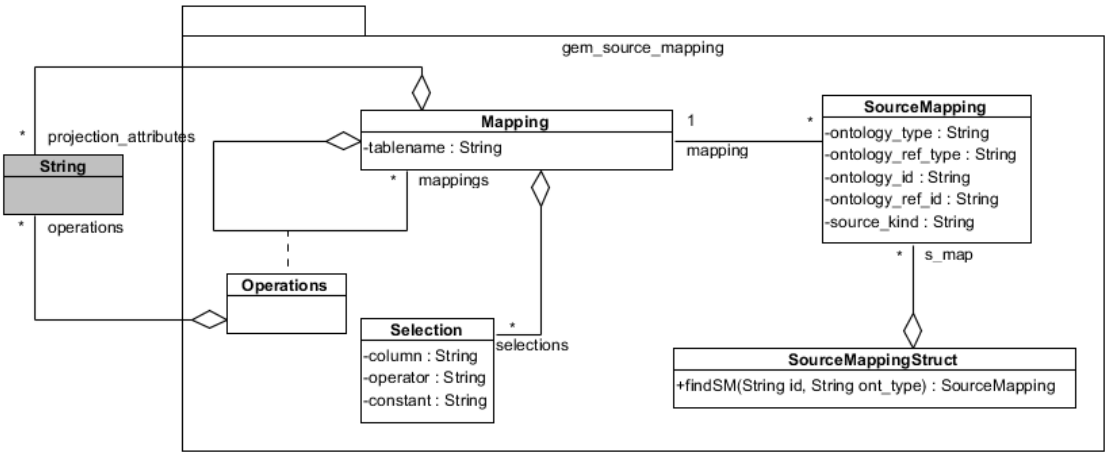


Figure 14: Part of class diagram representing structures for source mappings

This design is driven by the XML specification of the source mappings, discussed in 3.2.1.3. The central class is *SourceMappingStruct* that contains all information extracted from the XML structure. It can be noticed that *SourceMapping* class contains information gathered from the first part of one source mapping, i.e., information about ontology types and identifiers. Class *Mapping* contains specific information about how this concept from the ontology is mapped to the real sources. It can be noticed that class *Mapping* can include the additional set of mappings. This comes from the fact that one ontology concept can be mapped to the sources through more than one mapping (derived mapping) including operators that should be applied among those mappings (*Operators*). Class *Selection* contains information from the selection part of the input XML and that is intended to limit the resulting set of corresponding mapping.

At the end of the design process, the design of the part responsible for handling the input ontology is considered. Even though OWL represents the most complex structure, design of this part happens to be very simple, since JENA library already contains all the structures for storing of the ontology elements. These internal structures can be later accessed through the corresponding interface that JENA offers. In the Figure 15 the simple design of this part is depicted. However, JENA library sometimes requires quite complex code for obtaining knowledge from the ontology. More details about these issues will be covered in the implementation parts of the following iterations.

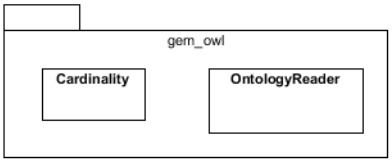


Figure 15: Package for reading the input OWL ontology

Implementation

Implementation part of this iteration starts with the development of the structures that will be used for storing the data extracted from the inputs. Following the previous design models and the structure of the input files discussed in section 3.1.1., programming of the structural part, is mostly straightforward.

Prior the implementation of the part responsible for reading the input XML structures, the definitions of these structures should be provided. For this purpose, the DTD files are defined. These files are used to define legal building blocks of the XML documents, i.e., document structure with a list of legal elements and attributes. Therefore, DTD files are provided for both business requirements and source mappings. The graphical representations of these DTDs are provided in figure 8 and figure 9 and the original DTDs are provided in Appendix B.

Concerning the ***business requirements*** input, for each section of the input XML structure, separated with the tags, the Java class is created.

The following classes, that represent the translations of the previous design and that are actually used for storing the content of the XML file, are created:

- *Concept.java*, with its subclasses:
    - *Level.java*
    - *Descriptor.java*
    - *Measure.java*

    These classes are created for storing the data about different concepts. These are mostly simple classes containing only corresponding the attributes and get/set methods. The only class that has some additional functionality is the class *Measure.java*. Since the measure concept contains the function for calculating the measure value, corresponding class (*Measure.java*) contains the method for extracting ontology concepts (*functionConcepts*) from this function. This is needed because the concepts included in the function are indeed ontology concepts that should be mapped and than included in the generation of the output designs.

- *NFR.java* with *AdditionalNFRInformation.java*, is used for storing the non-functional requirements that are related to a surrounding concept in the XML structure. The *AdditionalNFRInformation.java* class is used for storing the attributes and its values defined inside the *nfr* tag (e.g. format="HH24:MM:SS" in the <nfr kind="freshness" format="HH24:MM:SS" ><00:10:00</nfr>), since these attributes are specific for the particular non-functional requirement.

- *Aggregation.java*, is used for storing the data extracted from the aggregation part of the input XML structure. (<aggregations>)

After these structures are created, the part responsible for parsing of the input XML, extracting concepts from the business requirements and storing those concepts into prepared structures, is implemented.
Prior the beginning of this part of implementation, the specifics of the JAXP library are considered. The SAX API from JAXP has been chosen. This API, as already stated, parse the input XML via handling the events that occur during the reading process. These events are: *beginning of the XML document*, *end of the XML document, beginning of the XML element* (start XML tag), *end of the XML element* (end XML tag), *characters that appear between tags* and *whitespace that is actually ignored*. SAX API defines class *DefaultHandler* that represents basic, dummy handler of the input XML structure. However, this class does not offer useful means for XML parsing and therefore the programmer is required to extend *DefaultHandler* and override methods that represent the handlers for each specific event stated above. For that purpose, class *XMLHandler.java*, that extends *DefaultHandler*, is created, and corresponding methods are overridden, i.e., *startDocument*, *endDocument*, *startElement*, *endElement* and *characters*. Additionally the methods for handling the errors that are identified in the input XML file are also overridden. These methods are used for representing of the errors detected during the process of XML parsing, either because of the invalid XML format or because of the illegal structure that does not respect defined DTD.

XMLHandler class contains reference to the object of the *XMLStruct* class, because handler class is responsible for storing data that is read through the overridden methods. As already mentioned in the section covering design, *XMLReader* represents the class which main purpose is to prepare the structures needed for the parsing process. Therefore, this class, i.e., its method *readXML*, besides *XMLStruct* also prepares the object of the *SAXParser*, the class from the JAXP library, which actually performs the process of parsing. *SAXParser* object is obtained from the instance of *SAXParserFactory*. Afterwards, the instance of the *XMLHandler* is also created and the process of XML parsing is started by the invocation of the *parse()* method of *SAXParser* with the file that contains XML (*File*) and the instance of *XMLHandler* as arguments. The method *read()* from the *XMLReader* class receives the path of the file that contains XML and than prepares all the structures and the handler, parses the document and as the output it produces the instance of the *XMLStruct* filled with the information gathered in the parsing process.

Reading of the source mapping is quite similar to the one of the business requirements. The only difference is the structure for storing the content of the source mappings. Classes *SourceMapping.java, Mapping.java* and *Selection.java* are mostly simple Java classes with

corresponding attributes and get/set methods. This structure is already explained and depicted in the Figure 14, and therefore no further details will be considered.

In the Figure 15, it can be seen that part of system for reading the input ontology is not that complex. Reason for this is that JENA library, which is used for this purpose, already contains internal structures for storing data read from the input file.

Class *OntologyReader.java* is created with the main purpose to read input OWL file. This class requires the path to the OWL file and at the end it provides interface for accessing the structures read from the input. This interface, which actually JENA provides through the object of the OntModel class, is later used for accessing the elements inside the ontology. After the instance of the *OntologyReader* class is created with the name of the OWL file, the method *read()* is then invoked. This method creates new instance of the ontology model through the JENA's *ModelFactory* class. As an argument for the creation, the *OntModelSpec.OWL_MEM* parameter is provided. This parameter is used to define the kind of the ontology that JENA should expect at the input. After the instance of the appropriate model is created, it is later used for reading the input file, with the invocation of its *read()*. Ontology model created in this way does not contain any ontology reasoner, attach to it. Therefore, additionally the appropriate reasoning engine is added to the model.

Another class of this package is *Cardinality.java* that is used to represent cardinalities of the associations from the ontology. This is the point, already discussed, when JENA requires more complex way for obtaining, in this case, information about the association cardinalities. Cardinalities in OWL can be defined in several ways.

1. Through defining property (association) to be functional (*FunctionalProperty*), that actually means that the property's minimum cardinality is zero and its maximum cardinality is one
2. Through the restrictions inside the class definition (*minCardinality*, *maxCardinality* and *cardinality*). These restrictions are defined as subclasses of the class that they are applying on.

Because of this, unique method *getCadinality* is defined in the *OntologyReader.* This method takes two arguments, ontology class and ontology property (concept and association) and using the above possible definitions tries to calculate the minimum and the maximum cardinality of the given concept in the given association. Method returns the instance of the class Cardinality that is mentioned above, filled with the calculated values.

Obstacles and solutions

As already mentioned, at the beginning of the implementation part of this iteration, the definitions of the XML documents should be provided. However, since this is the research

work, during this iteration the structures of the input XML files have changed and also their DTDs, in order to support some new ideas that arose in the way. Along with these changes it was necessary to change the structures prepared as the storage for the XML content and the methods prepared for handling different XML elements. Fortunately, the original code was prepared in a way to be feasible to efficiently support quick adaption to the later structural changes. If new attribute or child tag is introduced in the XML structure, it will only be necessary to add a new attribute into the corresponding structure and, in case of a new tag, a new flag inside the *Flags* class. Additionally, the changes of the appropriate methods of the *XMLHandler* class are also minor and includes introduction of a new conditional block that handles new element.

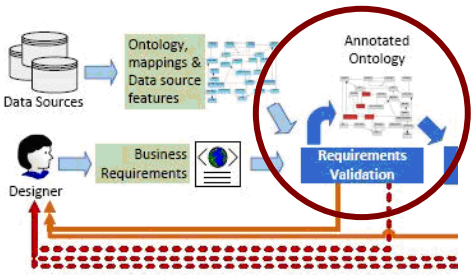
Another issue that I faced during this iteration was the function of the measure concept. This function included the concepts and different arithmetic operators (+, -, /, *) that are used to calculate the value of a measure. It was discussed that for the sake of the conceptual MD and ETL designs only the concepts included in the function are actually important. Therefore, afterwards, we agreed that the concepts from the function need to be extracted and included in the process of producing the output designs. However, since the possible extensions of this work may include moving from the conceptual design level to the logical and physical levels, where the information about this function is actually important, it was agreed to somehow forward this function as a whole in order for another design levels to be able to access it. The result of this was the introduction of the method *extractFunctionConcept*. This method deals with the problem of extraction of the necessary concepts but it does not consider the actual semantics of the function. After the concepts are extracted they are stored in the structure inside the *Measure* object, and the function as a whole is also stored and available to be forwarded to another level of MD and ETL design. Similarly to this issue, non-functional requirements, extracted from the input business requirements were not processed at this level, but are stored inside the *nfr* package and available to be forwarded in the future to another design level.

Results of the iteration

Following the starting requirements during this iteration the subsystem for reading, parsing and storing of the input XML and OWL structures is fully designed. As the main result of this iteration, the functional module that fulfills the requirements has been produced. This module as an input receives three different files, two XML files (*business requirements* and *source mappings*) and one OWL ontology file (ontology that represents *source data stores*). For storing the content of the input files, the corresponding structures are provided. These structures at the end of the parsing process can be obtained for their later exploitation.

4.3.2. *2nd iteration* – Requirement Validation

Requirements



After the functionalities for handling the inputs are implemented, the requirements for the realization of the first stage of the GEM system are considered. The first stage of the GEM framework is Requirement Validation stage. Requirements for this iteration are mostly gathered from the specification of the GEM framework in chapter 3. However, they are listed here as well, since they drive this iteration:

- Considering the inputs, the system should be able to identify each concept from the business requirements inside of the available source data stores (ontology).
- Afterwards, the system should tag every identified concept of the ontology with the corresponding label (Level, Measure or Descriptor), depending on the business requirement file.
- Then, for every tagged concept the system should identify the mapping of that concept to the sources. This part is explained in more details in section 3.3., where the whole stage of Requirement Validation is explained.
- During the mapping process, if the concept cannot be mapped directly, user might be asked to provide two kinds of feedbacks:
    - New derived source mapping according to proposed suggestions, and
    - The chosen synonym from the proposed list.
- As the result, requirement validation stage should provide the system with the subset of the ontology concepts annotated according to the input business requirements

This set of these requirements is expressed by the use case diagram depicted in the Figure 16.

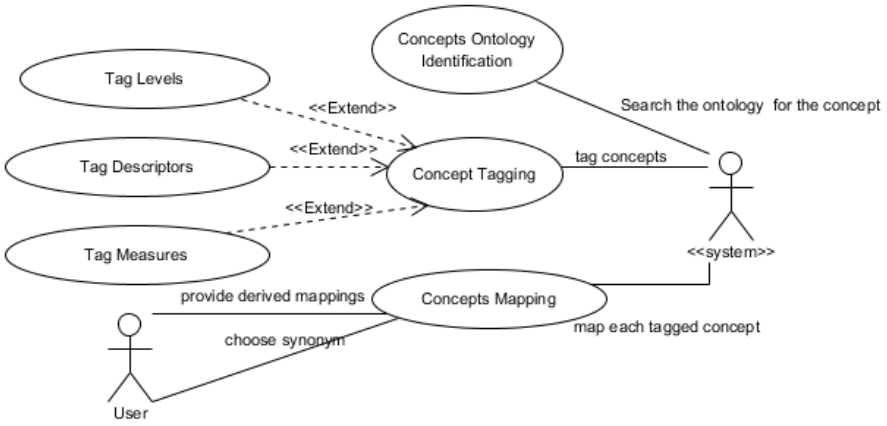


Figure 16: Use case diagram of the Requirement Validation module

Design

To respond to the stated requirements, appropriate design of the resulting Requirement Validation module, is provided. The basis of this design is certainly corresponding class diagram and appropriate sequence diagram that depict the process of requirement validation. In the given class diagram (Figure 17.) the central class is *GEMRequirementValidation*. This class represents the interface of the requirement validation part for the rest of the system. It contains three attributes.

These attributes are actually the structures that contain input data discussed in the previous iteration. The operation that should model the main functionality of the validation process is *requirementValidation().* The first two use cases in the use case diagram, i.e., *identification* and *tagging of the concepts*, are modeled with the package *gem_concept_tagging.* This package contains three classes, more specifically two classes (*TaggedConcepts* and *Tag*) and one enumeration (*TagOption*). *Tag* represents the relation between the concept extracted from the business requirements (*concept:Concept*) and the corresponding ontology concept (*ontResources*).

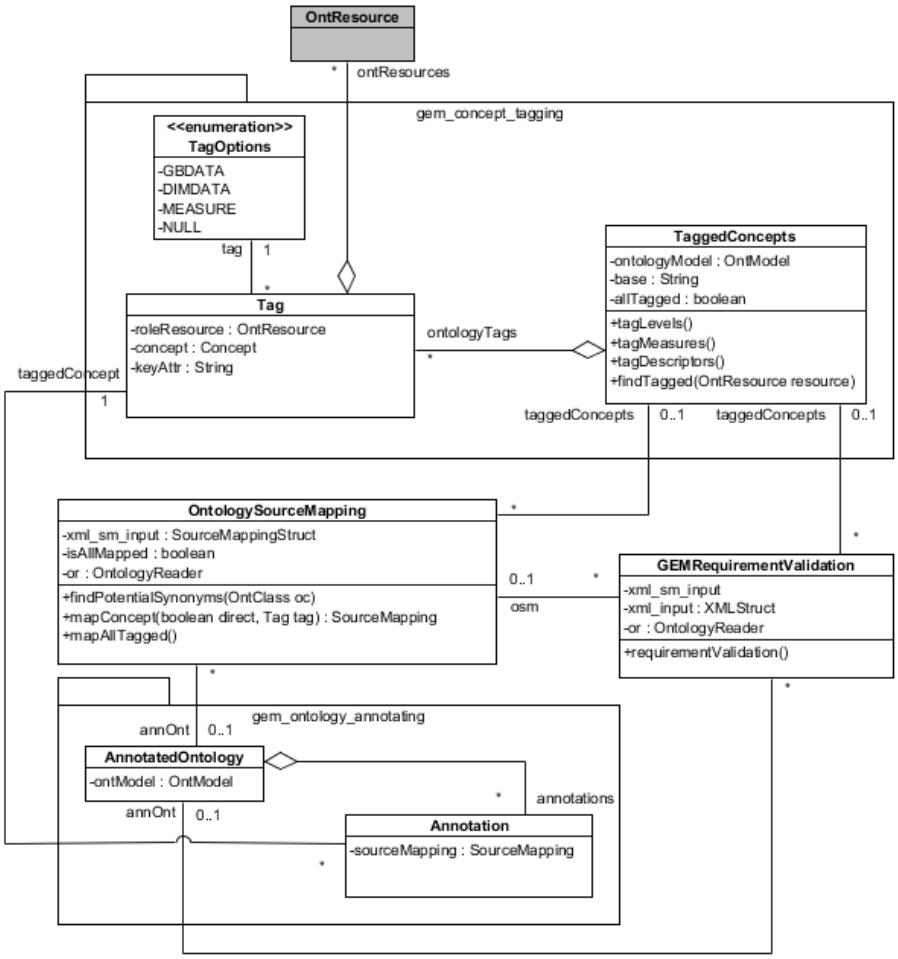


Figure 17: Class diagram for Requirement Validation module

OntResource is JENA's class that is used to generally represent an element in the ontology. More details about this are provided in the section concerning the implementation. It can be noticed that one concept from the business requirements can be related to more than one corresponding ontology concept. Reason for this is already mentioned case of the measure concept, where one concept is actually consisted of the set of concepts that are extracted from the measure function. Primarily through the methods tagLevels, tagDescriptors and tagMeasures the concepts from the business requirements are identified inside the ontology and tagged with the corresponding label, i.e., *TagOption (GBDATA* for Levels, *DIMDATA* for Descriptors and *MEASURE* for Measures). This is represented with the steps 1.1, 1.3, 1.5 of the sequence diagram (Figure 18). Afterwards, respecting the requirements expressed in the use case diagram, for each of these concepts the corresponding mapping to the source is searched.

This is modeled with the class *OntologySourceMapping* and its operations *mapAllTagged* and *mapConcept*. In the sequence diagram we can see that this is the step 1.7 (invocation of *mapAllTagged* operation).

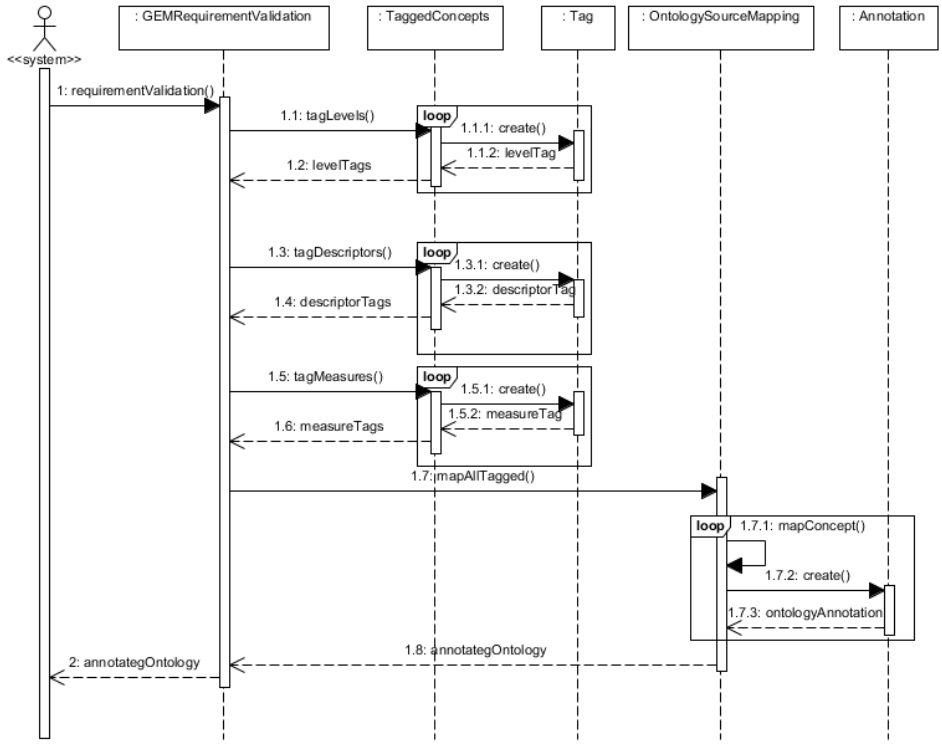


Figure 18: Sequence diagram for Requirement validation stage

Inside the *mapAllTagged* operation, for each tagged concept corresponding mapping should be found with *mapConcept* operation (step 1.7.1). During the mapping process, the set of the ontology annotations is created. For each tagged concept that is mapped to the sources the corresponding *Annotation* object is created (step 1.7.2) and added to this set.

Probably the most important and the most complex functionality inside the requirement validation part is the one that is responsible for mapping of the tagged concepts to the real source data stores (*mapConcept – step 1.7.1 in the sequence diagram*). The general algorithm for finding these mappings is presented in section 3.1.2.3. However, due to better understanding of this process it is depicted in the activity diagram in Figure 19. The input for this operation is the *Tag* that contains the concept previously identified in the ontology and tagged with the corresponding label. Afterwards, in the structure containing the source mappings, which is discussed in the previous iteration, the mapping for this tagged concept is searched. If the direct mapping is not found then the mapped subclasses are searched. If any mapped subclasses is found, the system suggest user to derive new mapping for original concept according to subclass concepts that are mapped and additional operations. Suggestion is proposed to user and the user feedback is then expected. If none of the subclasses is mapped than the system makes the same search but this time for the superclasses. If none of the superclasses is mapped that the system searches for possible synonyms of the concept. The system then offers the list of the potential synonyms to the user and expects the user response with the chosen synonym. If none of the synonyms is mapped then the system resets the process of requirement validation and proposes to user to update source mapping input structure.

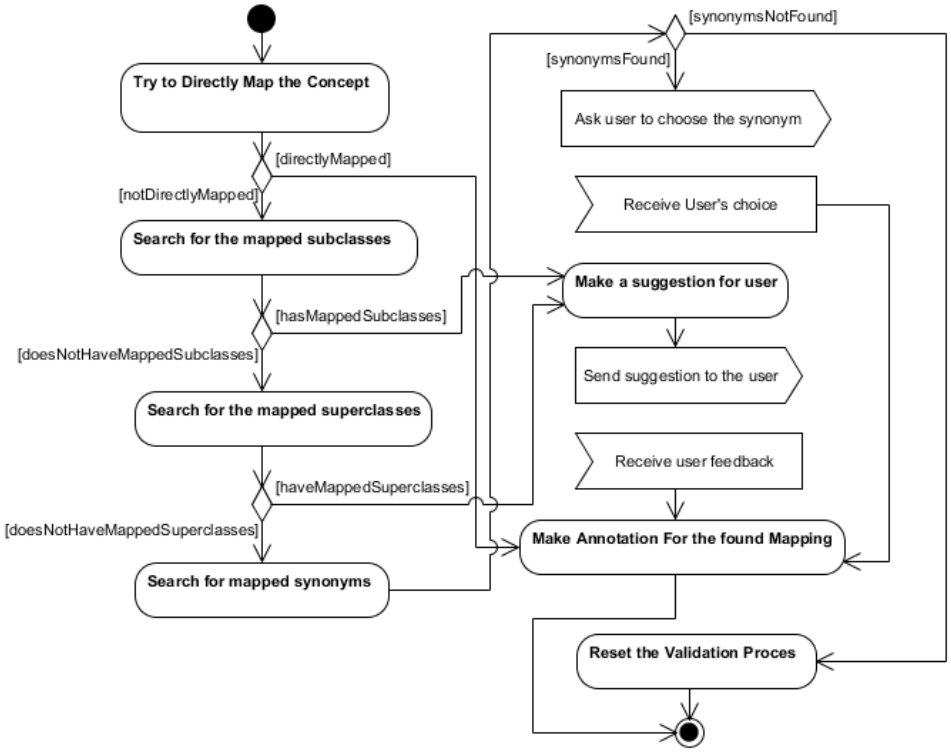


Figure 19: Activity diagram for mapConcept operation

Implementation

Implementation of the previously discussed design, first requires detailed comprehension of the JENA interface for accessing the ontology that is previously read. Detailed Java documentation and overview of JENA are available in [20] and [21], respectively. As already

mentioned, OntResource is JENA's interface made to represent and access general element in the ontology. OntResource is actually the superinterface of different interfaces that represent various OWL concepts that can be declared in the ontology. Some of its most important subinterfaces and those that are used during the implementation are:

- OntClass for representing ontology node (Class),
- OntProperty representing ontology association (Property),
- DatatypeProperty representing ontology association whose range values are datatype values (DatatypeProperty),
- ObjectProperty representing ontology associations between instances of two classes (ObjectProperty),
- Restriction, representing constraints that can be used inside Class declaration in the ontology (Restriction). It actually extends OntClass because all restrictions of one Class in the ontology are actually declared as subclasses of that Class (section 3.1.1.1.). Some important subinterfaces of the Restriction interface, for defining association cardinalities, are: MaxCardinalityRestriction, MinCardinalityRestriction and CardinalityRestriction.

All the above interfaces contain methods for the access to different features of these ontology elements. Those methods are mentioned during the iteration phases where they are actually used.

Concerning the package *gem_concept_tagging* in Figure 17, the class *TaggedConcepts.java* is created. This class is used for realization of the first and the second requirement (*concept identification* and *concept tagging*). Required functionalities are implemented inside the methods *tagLevels, tagDescriptors* and *tagMeasures*. Inside all three methods concepts from the business requirements are first search in the input ontology. This search is implemented through the JENA's interface to the entire ontology (*OntModel*) with the method *getOntResource* that receives the String argument which is indeed ontology URI of that concept. URIs in the ontology represent unique identifier of the concept and besides the local name of the concept it contains base URI that is defined for the whole ontology. For creating of this URI it is necessary first to obtain base URI from the model and then to add the concept name that is gathered from the input requirements.

For example:
Base URI for the entire TPC-H ontology is defined as:

xml:base="**http://www.owl-ontologies.com/unnamed.owl#**"

If the name of the concept from the business requirements is:

Region_r_nameATRIBUT

Then the URI of this concept in the ontology is:

http://www.owl-ontologies.com/unnamed.owl#Region_r_nameATRIBUT

Moreover, the concept of the business requirement can be identified as either Class or DatatypeProperty in the ontology. However, both of them can be saved in the OntResource variable since it is their superslass. For later usage of this object, JENA prepared methods to check the real type of the ontology resource, i.e., *isClass()*, *isDatatypeProperty()*, *isProperty()* or the general one canAs(*OntResourceSubType.class*) that can be use for any type of the ontology resource. Additionally, JENA also prepared methods for conversion from the OntResource to these subtypes, i.e., *asClass(), asDatatypeProperty()* or the general one *as (OntResourceSubType.class)* that can be used for any subtype of ontology resource.

If the concept is identified in the ontology, then the instance of the *Tag* class is created. This instance contains the reference to the concept of business requirements (*Concept*), reference to the concept in the ontology (*OntResource*) and the label (*TagOption*) that depends on the business requirements.

Concerning the specificity of the *Measure* concept, the method *tagMeasures* differs from other two tagging methods. The difference is that, in this case the concepts extracted from the measure function are identified in the ontology and the resulting *Tag* instance will contain the set of references to ontology concepts, one for each extracted concept, but only one reference to the concept of the business requirements (one that is created for that *Measure* concept) and one label (MEASURE). As the result, after these tagging methods, the set of the tagged concepts is available (*taggedConcepts*).

After the concepts from business requirements are identified and tagged inside the ontology, the process of their *mapping to the source data stores* follows. This process is implemented in the method *mapAllTagged*() that takes the set of the concepts previously tagged and for each one invoke the method *mapConcept*().

This method is roughly designed with the activity diagram in the Figure 19. Following this design and respecting JENA specificities the implementation of this method can be described as follows.

The precondition for this method is that the source mapping structure, read from the input XML file, is available (*SourceMappingStruct*). First the direct mapping for the concept is searched inside of this structure. Direct, means that inside the structure there is the element with the *ontology_ID* attribute (Figure 14) equals to the ontology URI of this concept. If the concept cannot be directly mapped to the sources then we follow the algorithm represented in Figure 19. In the sequel, the implementation of this algorithm is explained.

For searching the subclasses and superclasses JENA prepared methods in the OntClass interface *listSubClasses*() and *listSuperClasses*(), respectively. Both of these methods can receive boolean argument that states if only the direct subclasses/superclasses (true) or all possible subclasses/superclasses (false) are searched. If the argument is not specified then the default value is false and this case is actually used here. When the mapped subclasses/ superclasses are found then the suggestion for the user is prepared. This suggestion contains found mapped classes and operators that are needed for derivation of the new mappings. For subclasses the only operator is UNION and for the superclasses if ther is more than one INTERSECTION or MINUS and if there is only one mapped superclass SELECTION. User then should derive new mapping according to the suggestion and load it to the system. System then reads new file and tries to extract correct mapping. If it does not succeed than user is warned again with the same suggestion.

As already stated in the original algorithm, if none of the subclasses/superclasses are mapped this method should search for potential synonyms of this class. Synonyms of the given ontology class are those ontology classes related to the first one by the association that has cardinality 1-1. Here the transitivity rule should be also applied. This means that the path with 1-1 cardinality associations should be followed to find all potential synonyms. Searching for the potential synonyms is implemented in the method *findPotentialSynonyms*(). This method as an argument receives the OntClass object which synonyms should be searched. This method uses the already discussed method *getCardinality* in the *OntologyReader* class to obtain the cardinalities of the properties of the given concept. First the mapping for all the classes that are related to the given class with the 1-1 association is searched, then the same search is recursively repeated for each found synonym, until all the possible synonyms are identified. At the end the set of the potential synonyms is returned. Then the system prepares this list and presents it to the user. The system then waits for the user to make his/her choice after which it continues. No matter what kind of mapping is found (direct, through subclasses/superclasses or through the chosen synonym) the instance of the *Annotation* class is created. This object contains the instance of the *Tag* class previously made for this concept and the instance of the *SourceMapping* class that represents source mapping found for this concept. This annotation is then added to the list of the ontology annotations (*AnnotatedOntology*). Instance of the *AnnotatedOntology* actually represents the final output of this module.

Obstacles and Solutions

The main issue of this iteration was the derivation of the concept's mapping in the case that the concept is not directly mapped. The ambiguity that may arise and the appropriate solutions for this problem are already explained in section 3.1.2.1., and thus are followed in this iteration.

Concerning this mapping process another issue has arisen, now due to implementation limits. Since the tagged concept can be either *Class* or *DatatypeProperty*, the issue of applying the given algorithm to the *DatatypeProperty* arose. In fact, in JENA it is not straightforward to search class taxonomy for a *DatatypeProperty*. This problem is solved in the way that if the concept (Class or DatatypeProperty) is not directly mapped then if it is DatatypePropery the rest of the algorithm is actually not applied directly to that DatatypeProperty but to its domain class.

For example: If the direct mapping for the concept Customer_c_nameATRIBUT is not available, then the suggestion for the derived mapping is made according to the subclasses of the concept's domain class – *Customer.* Therefore, classes LegalEntity and Individual are considered for creating the derived mapping.

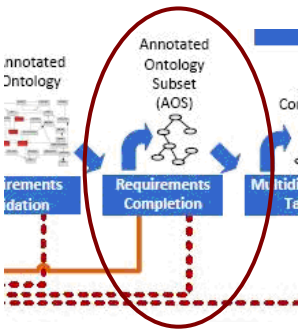
Another issue that arose in this iteration is the communication with the outside user. This communication is needed for giving suggestions to the user and receiving feedbacks from the user. Since this iteration mainly focuses on the implementation of the functionalities inside the requirement validation stage, this issue is temporarily solved through the standard input/output. However, one of the following iterations that is driven with the requirements for the Graphical User Interface properly solves this problem.

Results of the iteration

After this iteration the GEM framework is upgraded with the functionality for validating the business requirements that are previously read from the input. This validation process includes the process of tagging the identified ontology concepts and mapping of each tagged concept to the sources. Validation considers business requirements, on the one side and the data sources represented by the ontology and source mapping structure, on the other side.

4.3.3. *3rd iteration* – Requirement Completion

Requirements



After the system, in the previous two iterations, is provided with the functionalities for reading of the inputs and validation of the input business requirements, in this iteration the system should be provided with the functionalities for completion of the validated business requirements. This functionality represents the second stage of the GEM framework and its complete description is provided in section 3.1.2.2. Nevertheless, in the sequel the list of the requirements that drive this iteration is provided:

- In order to complete the business requirements, the system should be able to, according to the validated requirements (annotated ontology form the previous iteration), find the paths between the tagged concepts, in the ontology.
- In order to find these paths the system should identify intermediate concepts that are not originally required but are necessary to obtain the ones originally required.
- Path between tagged concepts should contain only mapped ontology elements (Classes and Properties)
- Associations (Properties) on these paths should respect summarization conditions (not many-to-many associations) discussed in [6]
- If between two tagged concepts there is more than one different path then the user should choose one that best fits the business needs.
- As the result, the system should, according to the paths, produce the graph containing nodes for both tagged and intermediate concepts and the edges, relating these nodes, which represents the ontology associations.

This set of these requirements is expressed by the use case diagram depicted in the Figure 20.

Technologies

Through this iteration, new technologies have not been introduced. However, JENA functionalities for accessing the ontology structure are also used in this iteration.

Design

In order to answer the requirements depicted in the use case diagram, the design of this part of the GEM system should be provided. For this purpose the class diagram in figure 21 is provided.

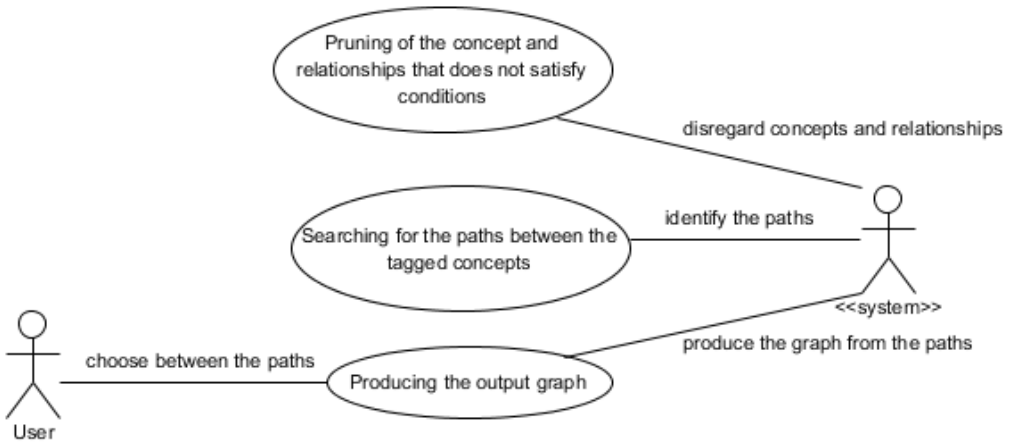


Figure 20: Use case diagram of Requirement Completion part

The package that models the requirement completion process is gem_requirement_completion. The central class of this diagram is

GEMRequirementCompletion and its operations *requirementCompletion* and *producingOutputPath.* This class actually represents the interface of this part to the rest of the system. Besides this, class Path models a path between tagged concepts. At this point, class MGraph will be also introduced for the first time.

As already mentioned, **Multidimensional Validation** stage of GEM framework has been already implemented. The MDBE system, that is the result of professor Oscar Romero, and his PhD work, covers the stage of Multidimensional Validation but also the previous process of Multidimensional Tagging. The central concept of the MDBE system is the MGraph class. This class is actually the graph consisted of the ontology concepts (tagged and intermediate) and relations between those concepts. After the stage of Multidimensional Validation this graph represents the MD design needed to retrieve the required information.

The requirement that has arose in this iteration is that the main output of the Requirement Completion stage should actually be the object of the MGraph filled with the identified paths between tagged concepts. Therefore, the MGraph is included as an external concept in this design.

The *requirementCompletion* operation actually covers the first two use cases

- *Pruning of the concepts and relationships that do not fulfill conditions,* and
- *Searching for the paths between the tagged concepts*.

The *requirementCompletion* operation actually represents the algorithm described in section 3.1.2.2., with the additional pruning process that is here included in the algorithm. That means that during the exploration of the new paths, along the way, the non-mapped/non-tagged concepts are ignored together with the many-to-many relationships. In the first part only direct paths are searched.

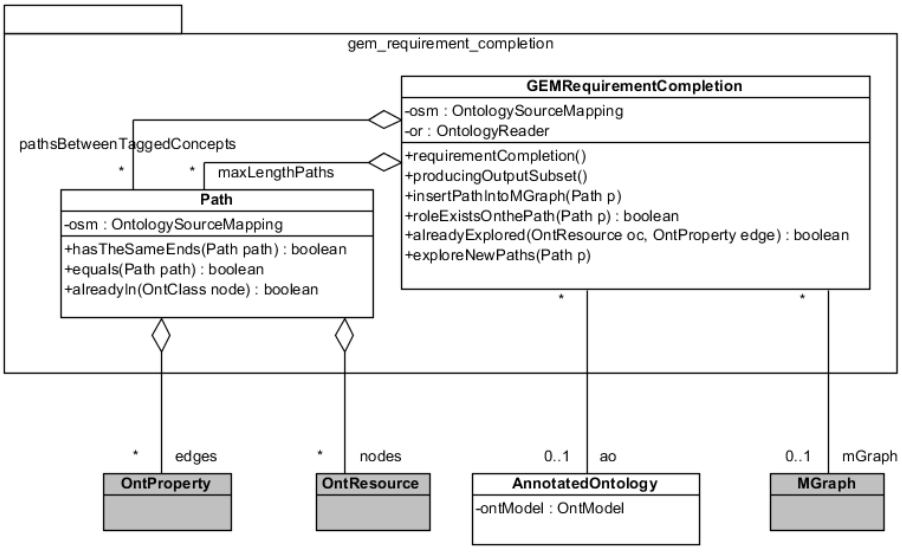


Figure 21: Class diagram of Requirement Completion module

This means that only the tagged concepts that are directly related in the ontology are added to the final set of paths. Additionally in the first part, for the concepts that are tagged but that does not have tagged neighbor, the path containing only one node (that concept) is created. Later in the second part of the algorithm, these paths with only one node are extended with new mapped associations and concepts. This process of path extension goes until the first tagged concept is reached. For each path between tagged concepts that are through the algorithm found in the ontology, the new object of *Path* class is created. After all the paths have been identified the operation *producingOutputSubset* should be invoked. This operation, as already stated, is used to build output graph out of the identified paths. Inside of this operation, if there is more than one path that has the same *seed* and *end* node, the user is asked to choose between those path one that best suits business needs. For designing of this operation specific needs of the already implemented MGraph should be considered. Since this represents the part of the integration process, more details about the MGraph is provided in the iterations related to the iteration of the MDBE system.

Implementation

The complete functionality of this part of the system is implemented inside the *gem_requirement_completion* package, more specifically with two classes *GEMRequirementCompletion.java* and *Path.java*.

Implementation part of this iteration starts with the implementation of the class *Path.java*, which is used for storing the paths between the tagged concepts that are identified in the ontology. After this structure is provided the algorithm that searches for the paths between tagged concepts is implemented. The method that implements mentioned algorithm is *requirementCompletion*. The input of the algorithm is, as already discussed, the annotated ontology containing the tagged concepts. For each tagged concept found in the annotated ontology in the first part the directly related concept that is tagged is searched.

At this place, the two new methods implemented for searching of the properties (associations) for a given concept should be introduced. Those methods are *findNewPropertiesDomain* and *findNewPropertiesRange.* Two methods are needed because one concept in the ontology can be related to another in two ways: As the domain of a property, or as the range of a property. In both methods, the properties, in which the given concept is included, are searched. JENA library offers the interface for accomplishing such thing. The method *listStatements* of the *OntModel* interface represents general way for obtaining various elements from the ontology that matches certain pattern. This pattern is defined through the input arguments of this method. For this purpose we need the following patterns (arguments): *listStatements(null, RDFS.domain, oc)* and *listStatements(null, RDFS.range, oc)*. This means that these methods will return the list of the ontology properties

where the oc (OntClass) is defined as the domain/range class of that property. Additionally for each property from this list it is checked that the property is mapped to the sources. For this purpose the already mentioned method of the *OntologySourceMapping* class, *mapConcept*, is used. As the method *mapConcept* previously supported only *OntClass* and *DatatypeProperty* types, now it has to be modified in a way to support the search for the mapping of the ontology property (*OntProperty*). Mapping of the properties between the concepts should be also found in the input XML structure with the source mappings. Furthermore, the mappings for the properties can only be search directly, i.e., there has to be a *SourceMapping* instance in the set of source mappings with the ontology_ID equals to the ontology URI of the given property. If the property is mapped than the additionally check is done. As already stated, associations with many-to-many cardinality should be ignored. Therefore, the system examines the cardinalities of the given property, using the method *getCardinality* from the class *OntologyReader.java,* discussed in the first iteration, and ignores those with many-to-many cardinality. After these methods return the list of corresponding mapped and not many-to-many properties, it needs to be checked that this property and the node on its other side has not been already explored. This is the way to guarantee that the path does not contain cycles.

These two methods (*findNewPropertiesDomain* and *findNewPropertiesRange*) are actually used in both first and second part of the algorithm. In the first part, after the mapped property from the given concept is found, the concept on the other side of the property is searched. Afterwards, the system checks if the found property is tagged. If it is, the new instance of the class *Path* is created and added to the *pathsBetweenTaggedConcepts* list. This list is actually the output of the whole paths-search algorithm. If the concept is not tagged, the system checks if it is mapped. If it is mapped than the instance of the Path class is created, but this time without end node, and is added to the *maxLengthPath* list. This lists represents the set of the path for which the exploration process is not finished, i.e., tagged concept on the end side of the path is not yet found. This list represents the input of the second part of the paths-search algorithm.

The second part of the algorithm, as already discussed, extends the previously found paths (*maxLengthPaths*). This extending process is actually the incremental search, following the same principles as one in the first part, where the new properties (that fulfill the above discussed conditions) are added to the path in each iteration, until on the other side of the property the tagged concepts is identified. Then the complete path, where both seed and end nodes are tagged is added to the list *pathBetweenTaggedConcepts*. After the method *requirementCompletion* is run and paths between tagged concepts are identified, the method *produceOutputSubset* is called to generate the final set of the paths and, according to this set, to create the *MGraph* structure. First for each path, the paths with the same seed and end nodes are searched. For this purpose, the method *hasTheSameEnds*() in the class

Path.java is implemented. If more than one path is found than the list of these paths are given to the user and the user is asked to choose one of them. For the sake of more friendly interaction with the user, the default choice is also provide as an option. The default choice is the first-shortest path that is found. The system waits for the user response and according to user's choice inserts the chosen path into the *MGraph* structure.

The method created for inserting the path into the *MGraph* (*insertPathIntoMGraph*) should be then called for each path. This method goes through the given path and the nodes from the paths are added one by one to the resulting *MGraph*. Since this is the point of integration of the already developed part of GEM (Multidimensional Validation), more details about this method will be provided in the iteration dedicated to the integration of Multidimensional Validation stage.

Obstacles and Solutions

One of the issues arose in this iteration is the pruning process. This process is in the original algorithm described as separated process that goes before the path search process. However, since JENA stores all the ontology elements (including properties) inside the internal structures, making the new subset of ontology elements with the pruning process did not make much sense, especially considering the memory issues. Therefore, the pruning process has been included in the path search algorithm in a way that, before the system decides to add a concept or the property to the path it checks that the concept/property is mapped and that the association satisfy mentioned conditions (not many-to-many).

In this iteration the issue of the interaction with the user also arose. However, as already mentioned in the previous iteration, the main requirements of these iterations preferably considered the inside functionality of the implemented stages. The interaction with the user is here also temporally solved through the standard input/output. The later iteration that is primarily driven with the requirements for the Graphical User Interface solves user interaction issue, more properly.

Results of the iteration

The main product of this iteration is the module of the GEM framework fully responsible for the Requirement Completion stage. This module as the input takes the annotated ontology with the tagged concepts, completes it and as the output it produces the MGraph structure that contains both tagged and intermediate concepts (as nodes) on the chosen paths and all the properties that relates these concepts (as edges). As this is the final stage of the GEM framework that is covered with my thesis, it also represents one of the points where the modules that I implemented should be integrated with the ones that are already implemented by professor Oscar Romero and Daniel Gil Gonzales.

4.3.4. *4th iteration* – Integration of the *MDBE* system

After the stages of the GEM framework that my master thesis covers are implemented, the requirements for integration of these stages with the already implemented stages, arose. First, the integration with MDBE system was considered, since the result of MDBE represents one of the input structures for the next stage of Operation Identification. As already mentioned, MDBE is the system implemented by professor Oscar Romero, during his PhD work. This system includes the processes of multidimensional tagging and multidimensional validation. Therefore, this system inside the GEM framework first realizes the final step of the Requirements Completion stage (Annotating the Ontology AOS) and the complete Multidimensional Validation stage (section 3.1.2.3).

Pre-conditions

The originally implemented MDBE system, expected at its input the SQL query and the relational SQL schema. Since my work tends to overcome the problem of specifying the exact technology (relational), the certain adjustments of the MDBE system were necessary.

These adjustments were aimed at enabling the MDBE to now receive the MGraph at its input. Originally, MDBE, starting from the SQL query, first built the initial MGraph and then did the rest of the multidimensional taggings and validations. Therefore, this adjustment was quite effortless in the sense that only the first step of building the initial MGraph from the SQL query was supposed to be skipped, since the initial MGraph is already produced inside the Requirement Completion stage.

Requirements

When the process of integration with the MDBE system was considered, the team meeting was held. At this meeting the requirements for the process of integration were stated:

- MDBE, after the pre-conditional adjustments, requires the MGraph structure to be provided at its input. MGraph represents the graph structure and it should contain the following elements:
    - *Tagged nodes* that represent the tagged concepts, i.e., those concepts that are extracted from the input business requirements
    - *Intermediate nodes* that represent the concepts that are not tagged, but that exist on the path from one tagged concept to another, i.e., that are necessary for relating the tagged concepts in the ontology and thus are needed for retrieving the business required information.

- The naming of the nodes in the MGraph needs to correspond to the naming used when the ETLGraph is produces (in the Requirement Validation stage). This is, important because the stage of Operation Identification uses both ETLGraph and MGraph to generate the resulting ETL design.

Design

On the meeting held prior the integration process the design of the package that is relevant for the process of integration was presented by professor Oscar Romero. The design presented in Figure 22 represents only the subset of the package *mgraph* that is part of the MDBE system.

The central class in package *mgraph* is the *MGraph* class, which contains the resulting graph structure. This graph structure is represented with the set of *NodeInfo* objects that represents the nodes of the graph. *NodeInfo* object contains the set of edges *(edgeList)* that relate that node with the other nodes in the graph. Furthermore, the *Edge* class contains association towards the destination *NodeInfo* and multiplicities (Multiplicity) of that association.

Each *NodeInfo* object contains the Node object. This object provides more information about the particular graph node.

Information that are important for the process of integration, i.e., that needs to be filled after the Requirement Completion stage, are:

- *tablename* – name of the ontology concept
- *alias* - the alias of the ontology concept (to be discussed in the implementation part)
- Three instances of the *Attribute* class as follows:
    - *measures* – For the ontology concepts that are tagged as measures
    - *groupByData* – For the ontology concepts that are tagged as dimensions
    - *otherDimData* – For the ontology concepts that are tagged as descriptors
- *LabelOption* that represents the multidimensional role of the concept represented by that node.

After commenting the design of the MDBE system, now the changes of the design of the Requirements Completion part, for supporting the process of integration of MDBE, will be presented.

As stated, the part of the Requirements Completion design that supports the integration with MDBE is the operations *produceOutputGraph* and *insertPathIntoMGraph* of *GEMRequirementCompletion* class. Since the operation *produceOutputGraph* is discussed inside section 4.3.3., at this place the design of the operation *insertPathIntoMGraph* will be presented.

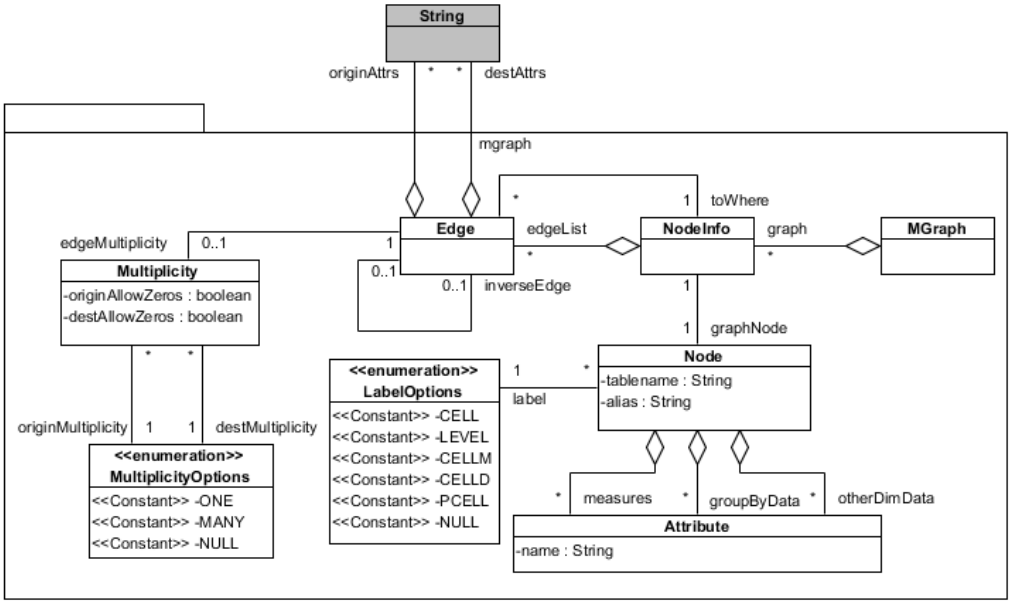


Figure 22: Part of the *mgraph* package from MDBE system (Oscar Romero)

In figure 23 the algorithm of inserting the path into MGraph is depicted. Input of the *insertPathIntoMGraph* operation is the object of the *Path* class created between two tagged concepts. This instance contains the list of the concepts through the path, and the list of the associations (properties) between those concepts. Therefore, the algorithm in figure 23 goes through this list and for each concept in the list, first searches if this node for this concept is already in the graph. If such node does not exist in the graph then the algorithm created new node and adds it to the graph. Then in both cases the algorithm adds the attribute that corresponds to the multidimensional role that a concept have. Afterwards, the algorithm relates the new added node to the node from the path that has been previously added. The outgoing edge of new added node is added to the graph, i.e., to the node's *edgeList.* For the end of the algorithm the multiplicities of the added edge are set.

Implementation

Implementation part of this iteration considers the implementation of the above discussed operation *insertPathIntoMGrap.* Implementation of this operation follows the algorithm depicted in figure 23.

The first part, where the system needs to check if the node for a given concept is already in the graph, is implemented through the invocation of the method *getNodeInfo* of the MGraph.

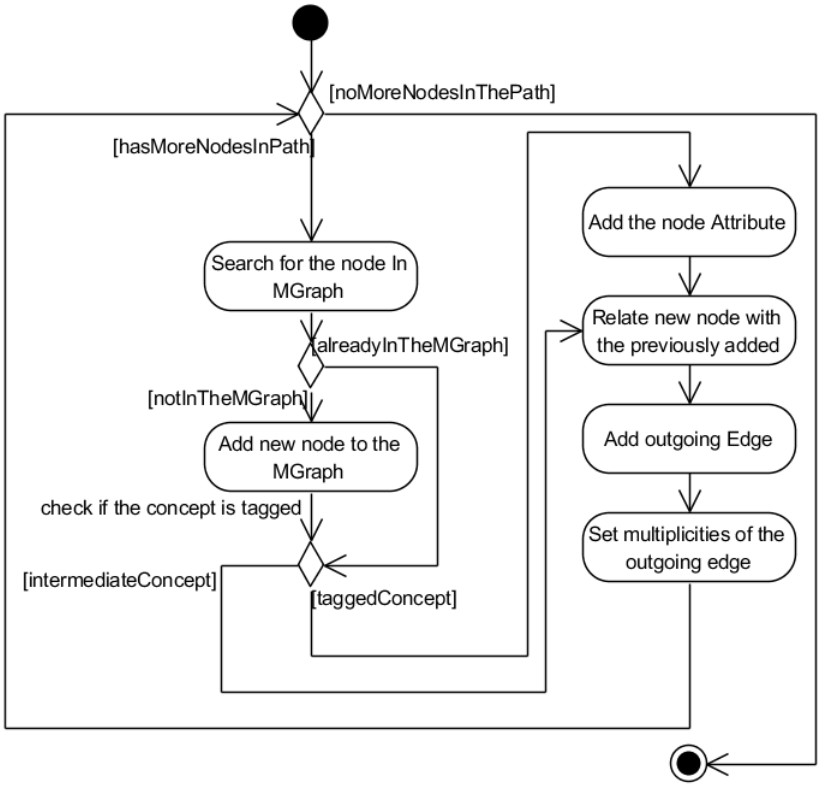


Figure 23: Algorithm for inserting the path into MGraph (*insertPathIntoMGraph*)

This method receives at the input the *tablename* and the *alias* of the node and searches the graph. If the node is found in the graph this method returns it, and if it is not the method return *null*. Then the *insertPathIntoMGrap* operation creates a new node. This new node should be created with the parameters *tablename* and *alias.* As explained, the names of the concepts in the MGraph need to correspond to the ones of the ETL graph, and therefore for the tablename the system uses the ontology URI of the concept. And for alias it uses the concepts alias if it is available and if not, again the ontology URI.

As it is explained, the concept can be identified as an OntClass or as a DatatypeProperty in the ontology. If the concept is identified as an OntClass the new *NodeInfo* is created with the ontology URI of that concept as tablename. However, if the concept is identified as a DatatypeProperty, then the system searches for the domain OntClass of that property, and created new NodeInfo with the URI of that domain class. Furthermore, the system uses this DatatypeProperty concept to fill the Attribute lists. The kind of list that will be filled (*measures*, *dimData*, *groupByData*) depends on a tag of the given concept and is explained in the design part.

After the outgoing edge is added, the system is supposed to obtain the multiplicities of the given edge. Therefore, the method explained in the 1st iteration, *getCardinality,* is used. This method is first called for the origin node, in order to retrieve its cardinality inside the edge. As arguments the system passes the OntProperty that represents the edge and the OntClass

that represents the node. At the same time, the system should obtain the destination node of the edge, which is easy since it goes through the list of all nodes on the path. For the OntClass that represents destination node and the same OntProperty the system again invokes the method *getCardinality.* When the cardinalities of both edge ends are obtained the system creates the instance of the *Multiplicity* class and sets its values according to the obtained ones.

After the appropriate structures are provided for the integration of the MDBE system the method *validate_requirement* of the *Mdbe* class should be invoked with the created MGraph as an argument. It should be noted that the method *validate_requirement* is the result of the MDBE modifications in order to support new input (MGraph). This method in fact, implements the process of multidimensional tagging and later multidimensional validation. Since for one input MGraph there can be more thank one combination of the intermediate nodes' taggings the result of the *validate_requirement* method (*MDBEResult*) can contain the set of MGraph structures, each one represents different combination of the tags. All of these structures can represent the output MD designs, and thus for each one of them the separate Operation Identification process should be started.

Results of the iteration

The main result of this iteration is that the MDBE system that was already implemented is now integrated with the fist two stages of the GEM framework. This indeed means that the MDBE system that was previously used SQL as its input format, now was enabled to receive the inputs provided in the general format (XML and OWL) and independent of any specific technology. This bridging between the MDBE and the initial GEM stages was succeeded by providing the necessary MGraph structure inside the Requirement Completion stage. This structure is built upon the concepts identified inside input business requirement and the paths between those concepts found in the sources.

4.3.5. 5th iteration – Integration of the Operation Identification stage

Requirements

After the first input for the Operation Identification stage is provided through the integration with MDBE (*MGraph*), the complete integration with the Operation Identification stage was considered.

Operation Identification stage is fully developed by Daniel Gil Gonzalez. After the team meeting the following requirements, concerning this integration, are stated:

- Integration with Operation Identification stage requires two structures to be provided as its input

    - EtlGraph – containing initial operation graph considering concepts' mappings
    - MGraph – representing final multidimensional design (result of the MDBE system)
- Therefore, besides the annotated ontology (tagged concepts) the Requirement Validation stage, at the output, should also produce the appropriate *EtlGraph*.
- *EtlGraph* at this point should contain the initial set necessary operations for the extraction of the tagged concepts from the sources.
- For each mapped concept from the input business requirements the subgraph in the *EtlGraph* should be provided.
- This subgraph should keep information about the mapping of the given concept to the source data stores
- The root of each subgraph should contain:
    - the name of the ontology concept that represents the source table to which the concept is mapped, and
    - the name of the ontology concept that represents the attribute of the source table that corresponds to the tagged concept, represented by the subgraph.
- These roots actually represent the points from which the complete EtlGraph is built inside the Operation Identification stage.
- Since this EtlGraph structure is later, inside the Operation Identification stage, complement according to the relations from the MGraph structure, the names of the source table concepts and source table attribute concepts should correspond to ones that appear in the MGraph. This is important and should be followed in the process of producing MGraph.
- The initial set of the operations, that should be provided in the Requirement Validation stage, depends on the mapping of the required concepts and it can contain:
    - Extraction – for the directly obtaining from the sources
    - Selection – for obtaining the concept from the sources using the discriminant function
    - Union, Intersection, Minus – for obtaining the concept from the sources through its mapped subclasses (Union) or supperclasses (Intersection or Minus).

Design

Design part of this iteration begins with the overview of the design of the ETLGraph package that was provided by Daniel Gil Gonzalez on the team meeting.

Since this design is not the part of my work, it is only briefly depicted with the main concepts that are important for the understanding of the integration process and that are actually needed to be provided before beginning of the Operation Identification stage.

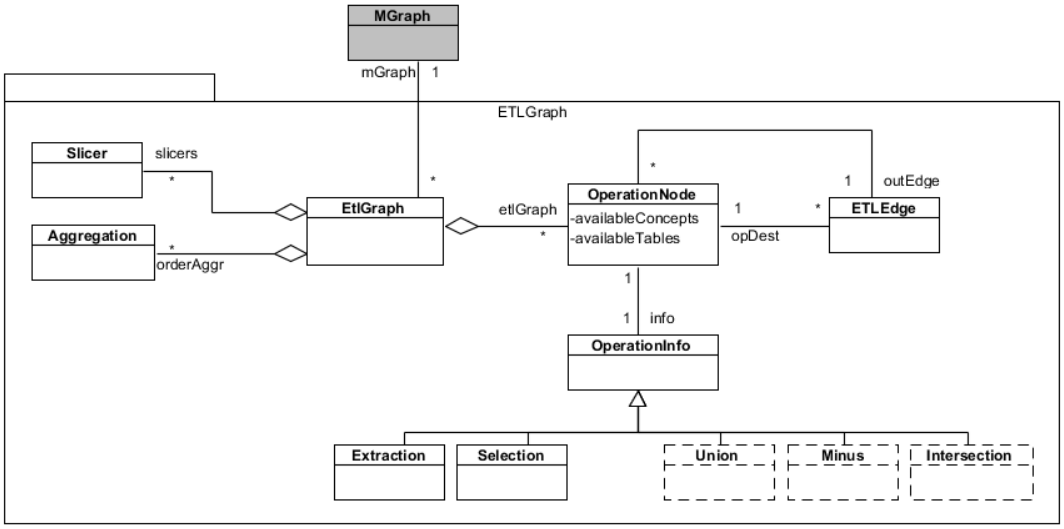


Figure 24: Class diagram of the ETLGraph package (Daniel Gonzalez)

Central class is *EtlGraph* and it represents the structure that should be initialized at the end of the Requirement Validation stage. As it can be seen, this class contains several structures.

- *mGraph* – this is the instance of the previously explained *MGraph* class. Since the instance of the *MGraph* is fully available after the Multidimensional Validation process then it is added to the previously prepared *etlGraph* structure
- *slicers* – that actually represent the list of the *Slicer* objects. *Slicer* represents the class for storing the information about the *Descriptors* (slicers) found inside the input business requirements file. This class corresponds to the *Descriptor* class from the *gem_xml.concepts* package (Figure 12).
- *orderAggr* – this structure stores the data about the aggregations found in the input business requirements file. This structure is the instance of the Aggregation class and it corresponds to the Aggregation class from the gem_xml.aggregations package (Figure 12).
- *etlGraph* – the last but the most important structure. This structure actually represents the ETL graph, i.e. output ETL design that should be generated as one the results of the GEM framework. It represents the set of the nodes that correspond to the operations (*OperationNode*). Besides other attributes, here are presented only those that are important for the integration process, i.e., whose values are provided in the previous stages of the framework.
    - The attribute *availableConcepts* represents the ontology concepts included by this operation and

- The attribute *availableTables* represents the source data stores from which these concepts are extracted.
- The *info* (*OperationInfo*) represents the structure that gives additional information about the operation represented with the particular node (*OperationNode*). This additional information is about the type of the operation that this node represents. Since the individual development of the Operation Identification stage did not consider mapping issue, the original set of the subclasses of the *OperationInfo* class (possible operations) was previously limited with the operation of Extraction, Selection, Projection, GroupBy and Join. After the team meeting and prior the integration, this set is complemented with the subclasses that represent the operations that are needed for mapping of the ontology concepts to the sources (Union, Intersection, Minus) and thus in Figure 24, they are depicted with the dashed line.

Another class important for the process of building the ETL graph is ETLEdge. This class represents the way of relating two nodes (OperationNode) inside the ETL graph. Therefore, this class is related with the OperationNode class with two different associations. First one depicts the relation of the node with its output edge, while the second one represents the relation of the edge with is destination node.

Since the design of the ETLGraph package is briefly overviewed, in the sequel, the upgrades to the original Requirement Validation design, made as a support for the integration process, are represented.

The only change to the previous design (Figure 17) is the introduction of two new operations in the class GEMRequirementValidation and the updated class is represented in Figure 25.

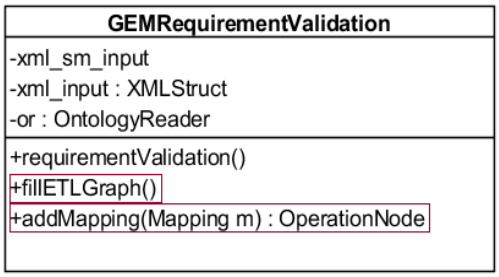


Figure 25: Changes to the previous design

The operation *fillETLGraph* should create new object of the *EtlGraph* and for every tagged concept it should invoke *addMapping* operation. The *addMapping* operation, takes one mapping that corresponds to one tagged concept, and according to that mapping it builds the subgraph inside the EtlGraph. The algorithm for building the subgraph is due to its recursive complexity depicted in Figure 26 with the corresponding pseudocode.

The algorithm in Figure 26 consists of two parts.

- The first part is responsible for creating the EXTRACTION node in case that the mapping of the concept is direct (line 6). Additionally if there is the discriminant function in the mapping (Selection tag) then the algorithm creates new operation node (SELECTION node) (line 14) and relates it to the corresponding EXTRACTION node with should represent its child node (line 16).
- The second part of the algorithm is responsible for the case where the mapping is not direct (derived mapping). Then for each direct mapping inside the derived one (mappings of the subclasses/superclasses) (line 25) the algorithm makes the recursive call with that direct mapping as an argument (line 27).

```
1   operation addMapping(mapping:Mapping): OperationNode
2     begin
3             OperationNode extraction, selection, temp, root;
4             if (mapping.isDirect())
5             begin
6                       extraction : = createExtractionNode(mapping.getTablename(),
                                        mapping.getProjAttrs,
                                          "EXTRACTION");
7                       etlGraph.add(extraction);
8                       root := extraction;
9                       if (mapping.containsSelection())
10                      begin
11                                temp := extraction;
12                                foreach (selection : mapping.getSelections())
13                                begin
14                                          selection :=
createSelectionNode(mapping.getSelectionAttr(),
mapping.getSelectionOperation(),
      mapping.getSelectionValue(),
                "SELECTION");
15                                        etlGraph.add(selection);
16                                        selection.addChildNode(temp);
17                                        temp := selection;
18                                end
19                                root := selection;
20                      end
21            end
22            else
23            begin
24                      OperationNode[] leaves;
25                      foreach (subMapping : mapping.getMappings())
26                      begin
27                                leaves.add(addMapping(subMapping));
28                      end
29                      temp := leaves(1);
30                      i := 1;
31                      foreach (operation : mapping.getOperations())
32                      begin
33                                temp := createOperationNode(leaves(i), leaves(i+1), operation);
34                                i := i+1;
35                                etlGraph.add(temp);
36                                temp.addChildNode(leaves(i));
37                                temp.addChildNode(leaves(i+1));

38                                leaves.set(i+1, temp);

39                      end
40                      root := temp;
41            end
42            return root;
43    end
```

Figure 26: Algorithm for building the ETL subgraph for a source mapping

This loop (lines 25 - 28) aims at providing the structure of the leaf nodes of the resulting graph. The terminology of the tree (leaf) is used because the resulting ETLGraph actually represents the binary tree because each node is supposed to have maximum two child nodes. This limitation is made in the implementation part of the ETLGraph and thus here inside the design of this algorithm needs to be considered. Afterwards, the algorithm goes through the loop (lines 31 - 39) and inside this loop, for each operation detected in the derived mapping it creates a node for that operation and relates it to its two child

nodes taken from the leaves list. As it is stated each node can have maximum two child nodes and therefore in (line 38) the algorithm adds the subgraph made out of one operation and two child nodes into the leaves list. This is important for respecting the implementation limits of the ETLGraph (maximum two child nodes), if the mapping is derived from the multiple direct mappings and operations.

Implementation

Implementation part of this iteration begins with the import of the corresponding package (*EtlGraph*). Afterwards, the implementation of the two operations presented in the design follows.

First, the algorithm for building the subgraph according to the mapping was implemented with adding the *addMapping* method inside the *GEMRequirementValidation.java* class. Implementation of this method mostly follows the algorithm depicted in Figure 26. However, the specificities of the ETLGraph, in order to build the correct graph, are respected. This considers the issue of naming the *availableTables* and *availableConcepts* attributes. As it is already mentioned, the *availableTables* should contains the names of the ontology concepts representing the source tables, and the *availableConcepts* should contain the ontology concepts representing the source table attributes that this concept is mapped to. As explained before, the concept in the ontology is uniquely identified with its URI, and therefore for filling of *availableTables* and *availableConcepts* attributes, the system uses the ontology URI of the particular concept. Regarding the naming issue, at the meeting it was agreed that since the implementation of Operation Identification stage considers only the roots of the subgraphs in the ETLGraph that are generated in Requirement Validation stage. The naming of the other nodes in the subgraph does not have to follow the same pattern. However, it is important to name the nodes in right manner for the later usage of the conceptual ETL design. Therefore, the availableTables attribute will be filled with the table names of the concepts' mappings included in this operation while the availableConcepts attribute will be filled with all the attributes' names from these mappings.

After the method *addMapping* is implemented the implementation of the method *fillETLGraph* is straightforward and it is consisted of the three parts.

- Providing the content of the *slicers* structure, from the content of the *Descriptor* objects of the *XMLStruct* (implementation part in section 4.3.1.).
- Providing the content of the *aggregations* structure, from the content of the *Aggregations* object of the *XMLStruct* (implementation part in section 4.3.1.).
- The invocation of the *addMapping* method, for each tagged concepts' mapping, with this mapping as an argument.

Furthermore, the appropriate methods of the ETLGraph, for building the resulting ETL design need to be invoked.

Obstacles and Solutions

The individually developed stage of Operation Identification, considered as an input only the structure resulting from the MDBE system. The problem was that the MDBE system was in fact, originally intended to work with the SQL queries and thus this limited the technology of input formats to relational. The great effort from both Daniel Gonzalez and my side was needed to overcome this issue during the integration of Operation Identification part into the GEM framework.

Some knowledge about relations between concepts that was previously extracted from the SQL queries is now supposed to be deduced from the underlying ontology. As an example, joins between data source tables are previously uniquely extracted from the WHERE clause of an SQL query and now they are supposed to be identified through the associations in the ontology. This can lead to the fact that some relations that are not necessary for the business needs are identified and thus included in the output design. It is noticed that this new relations may reveal some interesting analytical perspectives that the designer primarily had overlooked and thus they all should be considered. Therefore, the output design contains the superset of all the relations identified in the ontology, and it has to be adapted to the real business needs. After the discussion, this issue is solved by introducing new optional element of the input XML structure with the business requirements. This new element is *role*. The content of the *role* tags should be the concept to which surrounding concept should be somehow related. Therefore, the additional adaptations for resolving this issue included the modification of the path search process inside the requirement completion stage. After the path relating tagged concept to another tagged concept is identified, and if the tagged concept has the defined role, then the identified path is searched for the role concept. If the role concept is found in the path the path is accepted. Otherwise, the path is rejected since it does not relate the concept to its role concept. It is also noticed that the introduced solution tends to slightly automate the process of the path identification. This comes from the fact that if there is more than one path found between two nodes, the *role* may be the information that can filter some unnecessary solutions. Some additional explanations about the role concept and the examples are given in section 3.1.1.3.

Results of the iteration

The main result of this iteration represent the means of bridging the inputs of the GEM framework represent in a general way, i.e., not using any specific technology (e.g., relational), with the stage of Operation Identification. This bridging is possible since through the stages of the Requirement Validation, Requirement Completion and Multidimensional Validation the inputs are translated into the appropriate structures (ETLGraph and MGraph)

that Operation Identification stage requires and thus this stage starting from the initial set of the ETL operation detected according to concepts' mappings, and using the MD design provided with the MGraph, generates the final design of the appropriate ETL process.

4.3.6. *6th iteration* – Graphical User Interface

Requirements

As the previous iterations of the development process were concentrated on providing the main functionality of the system, this iteration as the final part aims at providing the means for the user to access the GEM system more easily and intuitively. Concerning the inputs and the points of the system where the interaction with the user is needed the following requirements arose:

- The graphical interface for the process of loading the input files is needed. Three different files at the input are required
    - OWL ontology representing data sources
    - XML file containing the mappings of the ontology concepts to the sources
    - XML file containing business requirements
- Since the first two files, i.e., OWL ontology and XML with mappings, represent sources it should be possible first to load these two files. Afterwards, considering already loaded source files, for each business requirement it should be possible to load new XML file. However, it should be also possible to reload the files representing sources.
- Inside the process of the concept mapping, if the concept is not mapped the user is asked to derive mapping according to the given suggestions. For this purpose appropriate graphical interface is required, where the suggestions will be first showed to the used and then the user should be able through the interface to load new XML file with the derived mapping
- Also inside the process of the source mapping, if none of the superclasses and subclasses are mapped then the system searches for the potential synonyms. When the list of the synonyms is provided the user should be asked to choose one concept that will be considered as the synonym. For this purpose the graphical interface, where the used can choose the synonym that best suits business needs, is required.
- Inside the Requirement Completion after all the paths between the tagged concepts are detected, if there are two or more different paths between the same concepts the user is asked to choose the one that he/she considers the most suitable. For this purpose, the graphical interface, where the user will be provided with the list of the paths and the possibility to choose one, is required
- After the stages of the Multidimesional Validation and Operation Identification the graphical representations of the produced multidimensional and ETL designs are required.

Technology

For the implementation of the graphical interface of the GEM system, ***Java Swing*** library was used. Swing represents the primary Java GUI widget toolkit. Swing is platform independent both in terms of expression (Java) and implementation (Look-and-Feel). From Java 1.2 edition Swing is included as a standard Java library, which made it even more accessible.

From the IDE point of view, as already mentioned, choice of using Net Beans as a development tool was partly led by the fact that Net Beans contains very intuitive GUI builder.

Implementation

After the internal functionality of the GEM system is implemented, another layer of GEM is supposed to be implemented, i.e. presentational layer. Implementation of the graphical interface, using Java Swing library, required creating the set of graphical elements that would realize the desired interface.

To group these graphical elements the package *gem_gui* is created. The main class of this of this package is *GEM.java.* This class extends the *JFrame* class from the swing library which means that it represents the window form that will be shown to the user.

After this, the class *GEMStart.java* is created (also extends *JFrame*) and it represents the starting point of the GEM process. From this frame the options for loading the different input files are provided.

To fulfill the first two requirements, i.e., loading of the input files, the frames containing the appropriate elements has to be created.

- For loading the business requirement the frame *GEMStart.java* is used. As for each loaded file, containing business requirements, the user should be able to start the GEM process, along with the loading element (*JFileChooser*) this frame also contains the option for starting the process.
- Afterwards, the frame containing the elements for loading OWL ontology and XML with mappings is created. *GEMLoadOWLAndMaps.java* is the class implementing this frame, while the elements responsible for loading the files are of type *JFileChooser* from the Java Swing package.

Furthermore, after all the files are loaded and GEM process is started the frame that follows the process is created. This frame (*UserFeedbackFrame.java*) contains the progress bar (*JProgressBar*) that shows the progress of the GEM process. This frame is also responsible for interaction with the user. Therefore, inside this frame the panel (*JPanel*) for containing the interaction elements is created. Since the interaction with the user depends on the execution process and the input files, this panel is supposed to be filled dynamically with the

appropriate elements. Therefore, three different methods for filling this panel are created inside the *GEMProcessFrame.java* class, one for each user interaction case. After the process is finished the panel for the interaction should contain the options to show the resulting designs. This is implemented with the additional method that according to the number of the resulting designs fill the panel with the buttons (JButton) for showing those designs to the user.
All these frames and different appearances of the user interaction panels are represented in figures of the Appendix A, where the demo of the GEM process is represented.

For achieving the last requirement, i.e., graphical presentation of the resulting designs, the credits go to Daniel Gil Gonzalez. Inside his project he used *JGraph* library to represents the results of his work. His work is slightly modified and adjusted to the GUI that I implemented for the GEM framework. Therefore, for each resulting design (MD or ETL) at the end of the process, the button for showing the design will start dynamical creation of a new frame and this frame will be filled with the *JGraph* panel. This *JGraph* panel contains graphical representation of the graphs (*MGraph* and *ETLGraph*) representing the output designs. The examples of these graphical representations of the resulting designs are also shown in figures 42 and 43 of Appendix A.

Results of the iteration

The main result of this iteration is the presentation layer of the GEM framework. This layer includes the graphical interface for:

- providing the framework with the necessary inputs,
- interaction with the user (designer) during the process of MD and ETL design generation,
- for representing the resulting designs to the user in the graph form.

4.4. Testing

The main goal of the software testing process is to ensure the quality of the developed system. Software testing is also the process for validation and verifying that the output software actually meets the business and technical requirements that guided its design and development and that it works as it is expected. During the process of testing the potential errors (software bugs) and other defects should be identified and accordingly resolved.

Testing of software may be, depending on a technique, introduced at various points during the development.

- *Unit (Modular) testing* that is supposed to be introduced for smaller units of the system (methods, classes, modules).

- *Integration testing* is introduced to verify the interfaces between different components integrated in a system.
- *System testing* tests a completely integrated system to verify that it meets its starting requirements.

Testing of the GEM framework developed as a technological part of this thesis relies on the TPC-H benchmark. As the development of GEM included integration of the already developed stages, besides unit and system testing, the integration testing is also included to ensure the communication among different modules is correct.

In the following sections, first more details about TPC-H benchmark are provided. Later the parts of this benchmark used for testing of GEM are introduced. Afterwards, three different kinds of tests that are used, is exhaustively explained.

4.4.1. TPC-H

The TPC-H Benchmark is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. As stated in [22], TPC-H benchmark is used as an illustration for the decision support systems that examines large volumes of data, executes queries with a high degree of complexity and gives answers to a critical business needs.

TPC-H is introduced for the testing of the GEM framework. However, since GEM for now considers schemas at the conceptual level, only some parts of TPC-H are used, and those are:

- The set of *data source* tables (source schema) depicted in Figure 28, and
- The set of *business queries* designed to test system functionalities with the real complex business analysis.

More information about TPC-H and its possibilities is provided in [22].

4.4.2. Adapting TPC-H schema for testing inputs

As explained, the GEM framework expects three different files at its input. For the purpose of testing of the GEM framework, these files need to be prepared. Since the TPC-H is introduced for the testing process the certain adjustments of its schema and queries are needed for providing valid inputs for the GEM system. Three different types of files and the corresponding TPC-H adjustments for providing these files are presented in the sequel.

1. **OWL ontology representing data source**

   Since TPC-H provides the set of relational data source tables, this set has to be appropriately transformed into the corresponding OWL ontology. However, as the original TPC-H source schema does not contain **taxonomies** and **one-to-one** relationships (synonyms), I extended this schema with the following concepts:

   - *LegalEntity* and *Individual* as subclasses of the concept *Customer*, modeling the fact that one customer can be either legal entity of an individual.
   - *Area* as a synonym of the concept *Region.*
   - After introducing concept *Area* it is also necessary to introduce the one-to-one association to *Region*

   These new concepts are intentionally introduced to be able to represent all the possibilities that GEM offers while searching for the concept mappings inside the requirements validation stage.

2. **XML file with the mappings of the ontology concepts to the sources**

   This file is created following the syntax explained in section 3.1.1.2. and considering the source schema. As the result we provide the following *SourceMapping* elements inside the mapping XML file:

   - First, one *SourceMapping* element is created for mapping of each ontology class, i.e. for mapping of its primary key.
   - Then, the mappings are provided for various datatype ontology properties that represent the concepts that can be found in the input business requirements
   - For ontology datatype properties that represents the references of one data source table to another (foreign keys) the appropriate mapping is provided.
   - At the end, the mappings for the ontology properties (associations) are created. Here intentionally we provide only some of the mappings to represent how GEM constructs different paths between concepts.

   Since TPC-H considers that the provided schema is relational, these *SourceMapping* elements are provided also with the *sourceKind* attribute having the value "relational".

3. **XML files representing different business requirements (queries)**

   After the structures that completely represent the source data stores (ontology and mappings) are provided, now we create several documents that represent the business queries according to which the final designs will be produced.

TPC-H provides us with the list of more then twenty general queries. Only the subset of those queries is used to test all the functionalities of the GEM framework. Previous the use, these queries needs to be translated to the XML form that is explained in section 3.1.1.3. The process of translation from the relational queries (SQL) to the XML format is given below:

- First, all names of the source table attributes needs to be adapted to the domain vocabulary, i.e., translated to the local names of the corresponding ontology concepts (without ontology URI).
- From the SELECT clause the attributes that are represented as measures are translated to measure concepts (<measures>), along with the functions for their calculation. In this process, since in relational queries the names may not be given to the measure attribute, the reasonable artificial name is generated for the measure concept.
- Then, the other attributes that appear in the SELECT clause, are supposed to be represented as dimensional concepts (<dimensions>).
- Considering the WHERE clause, if the expression from the WHERE clause represents the selection (slicer) than the descriptor (slicer) concept is created (<descriptors>).
- For each measure attribute and each attribute from the GROUPBY clause, we create one new <aggregation> element inside the <aggregations> section.

After the input test files are provided the process of the testing begun.

4.4.3. Unit (Modular) testing

Considering this part of testing different kinds of tests were prepared.

First, the JUnit tests are used to examine the correctness of the particular methods inside the developed classes. These tests are based on comparing the real values returned form the method with the prepared expected output values. For some methods it was difficult to provide appropriate JUnit test since their output and side effects were very complex. Therefore these methods are tested with the other kinds of tests, e.g., modular tests.

After each iteration, the set of modular tests were prepared. These tests are created for the verification of each developed module, following the explained guidelines. For the developed modules the tests prepared for each one of them are explained in the sequel.

1st iteration – testing of input reading and parsing

The main functionality of this part of the GEM system is the reading of the files from the inputs and their parsing into appropriate structures. Therefore, the tests prepared for this iteration aim at examining the correctness of these functionalities. Additionally the tests, that examine how the system reacts in cases that the provided input files are incorrect, are also provided.

All the tests considering the valid and correct input structures are produced from the TPC-H benchmark and following the previous guidelines. The content of the appropriate structures is compared with the expected values.

At this place while testing the larger XML files with the source mappings, one inconsistence is occurred. In fact, the content of some elements of the resulting structure (Source) was incomplete (e.g., instead "http://www.owl-ontologies.com/unnamed.owl#Nation" as a complete concept URI, only one part is stored "wl#Nation"). During the debugging process it is noticed that the problem occurs inside the method that handles the characters between tag elements (*characters()*). This method receives the buffer with characters and starting position in the buffer from where the characters should be read and the number of characters to be read. However, it is noticed that this buffer is limited to 256 characters. That means if the sequence of the characters is not finished by the end of the buffer, the buffer is restarted and the rest of the sequence is processed in the next invocation of the handler method. This resulted with the fact that only the end of the sequence, i.e. the part processed in the second invocation is stored. This is appropriately solved with the mechanism to concatenate the result form the one invocation and the next invocation of the same sequence.
As a result all prepared tests passed.

Considering the tests with incorrect inputs, several tests were prepared.

- Since XML files, as explained in [24], can be tested for correctness according to either ***well-formedness*** of XML elements and ***validity*** (based on the referenced DTD), two different kinds of test cases are prepared, to test the system reaction on these two irregularities.
    - *Malformed XML files* – the elements inside the XML are not formed correctly, e.g., tag elements are not nested appropriately, non-matching *begin* and *end* tags, overlapping of the tags, usage of the forbidden characters for tag names (!"#$%&)
    - *Invalid XML files* – the XML file disobey the definitions from the referenced DTD, e.g., missing the attributes that are defained to be required (#REQUIRED), missing the tag elements that are not defined as optional (?*).

- Considering the OWL ontology files, the incorrect files produced for testing contains the following incorrectness:
    - The concept referenced in the file is not defined in the same or any imported files
    - The definition of two concepts with the same ID (*rdf:ID*) exist.

The test cases that consider incorrect input files showed that the system appropriately reacts on such incorrectness. For XML files this is achieved through the error handler methods overridden in the *XMLHandler* and *SourceMappingXMLHandler* classes. On the other side, JENA already provides handlers for handling the errors inside the OWL files. These handlers and additional implementation provide system to react correctly on the errors in the input files. Therefore, if the incorrect input is provided the system do not continue with the process and the user is informed that the error is occurred with some additional details about the occurred error.

2nd iteration – testing of Requirement Validation stage

The module implementing the Requirement Validation stage at its input expects the structures prepared by the previous part after reading and parsing the input files. The main output of this module is the structure representing the ontology with annotations for the concepts identified in the input business requirements.

First the appropriate methods for tagging the concepts from the input business requirements are tested with the appropriate JUnit tests. These tests included input concepts (instances of *Concept* class), the OWL ontology where the concepts are searched, and for each input concept the expected output Tag object, containing the input concept and its corresponding ontology element (OntResource). As already mentioned, the output obtained after the method is invoked is compared with the expected output. These tests passed without problems.

Afterwards, the tests for testing the central method of this stage (mapConcept) should be prepared. Since this method is far more complex then the previous ones the JUnit tests are not used for testing its correctness.

For creating these tests the input file with the source mappings is considered. The tests for this method are made to examine all the possible cases of the concepts' mappings.

- Direct, found in the input file. For this kind of tests the concepts tagged in the input business requirements should have the corresponding direct source mapping in this file.

- Derived through the mapped subclasses and UNION operations. Here one concept is chosen to be tagged in the business requirements, but its direct mapping is not provided. However, the mapping of its subclasses is provided inside the mapping XML file. Additionally, the new file with the derived mapping is also provided, since the user will be asked to provide one according to the suggestions. The ontology classes chosen for this test case are *Customer* with its subclasses *LegalEntity* and *Individual.*

- Derived through the mapped superclasses and INTERSECTION, MINUS or SELECTION operation. The procedure for preparing these test cases is similar to the previous one, with the difference that in this case for the non mapped concept we provided mapping for its superclass(es) and the additional file with the derived mapping. The ontology classes for this test case are *LegalEntity* and its superclass *Customer.* Additionally, in the derived mapping the discriminant function is introduced (c_custkey > 2000000) which means that the keys for the customer that is LegalEntity must be greater than 2000000. Therefore the derived mapping is created form the mapping of the Customer concept and SELECT operation with the given discriminant function.

- Through the mapped synonym of that concept. For these kinds of test cases, for the given tagged concept the mapping is not provided, as well as for its subclasses/superclasses. The mapping is provided for the concept that is through the rule of transitivity related with the given concept with 1-1 association. The concepts that are used for this testing are the *Region* concept which is tagged and the *Area* concept which is mapped.

- Concept cannot be mapped. In this case we provide tagged concept and for one of the tagged we do not provide any of the above ways for mapping. The concept that is used for this testing is Customer but it is not mapped and neither are its subclasses. The results of this testing should show how the system reacts is some of the tagged concepts cannot be mapped to the sources. It is noticed that the system in this case stops the stage of requirement validation and warns user that it was not able to map all the concepts form the input business requirements. Additionally, the option of restarting the whole process is provided.

The above test cases that succeeded to map the tagged concepts produced at the output, the list of annotations (*Annotation*) that besides the *Tag* object for the given concept contains the *SourceMapping* structure. These structures are needed to be compared with the ones that were previously prepared as the expected.

The testing of the Requirement Validation module hasn't shown any grater errors and thus we continued with the development of the next iteration. However, at this place it should be stated that since the Requirement Validation stage has another structure at its output (ETLGraph) the correctness of this structure's content is tested inside the Integration test part.

3rd iteration – testing of Requirement Completion stage

This stage at its input expects the set of ontology annotations (*Annotation*) that the stage of Requirement Validation produces. The main output of this stage is the *MGrpah* structure that is used for integration with the *MDBE* system. Correctness of this structure will be discussed in the Integration testing part.

However, the structures that are tested after the 3rd iteration are from the set of the paths between tagged concepts (pathsBetweenTaggedConcepts), produced after the invocation of the requirementCompletion method in the *GEMRequirementCompletion* class and which are actually used for creating the *MGrpah* structure. This method is tested using the prepared JUnit tests. These tests first create the expected list of the paths, than the set of annotations produced in the previous stage is also provided. Then the method *requirementCompletion()* is invoked with this set of annotations and appropriate ontology. After the method ended, the content of the list that this method populated is compared with the expected list of the paths. This comparison is supported with the method *equals()* inside the Path class that examines the equality of two paths. The test cases for both direct and indirect paths between tagged concepts are created. Also the test case for the tagged concepts that cannot be related by any path is created.

All these tests passed with no major errors.

4.4.4. Integration testing

The first two modules that I developed are tested individually. In the phase after the iteration of Requirement Completion, the process of integration with the already developed modules starts (4th and 5th iterations). After the end of each integration iteration the appropriate *integration testing* is conducted.

This integration testing examines that the communication between the integrated parts is correct and that the communication interfaces are respected.

In this case the Bottom-Top testing is conducted, since I started from the testing of the individual modules and then after adding new one I conducted the integration testing for that integration.

I tested ***integration with the MDBE system*** first, because for testing the integration with the Operation Identification stage, the MGraph, fully tagged and validated, needs to be provided, and that is the main task of the MDBE system. Due to the complexity of MGraph structure and the lack of comparing method (equals()) the JUnit test are avoided for the method produceOutputGraph().

The testing of this integration process included preparation of the input files and performing first three parts, i.e., input reading and parsing, requirements validation and requirement completion, which are already tested. This will produce MGraph structure that is needed as an input for MDBE. Then the MDBE is started with the method *validate_requirement()* with the prepared MGraph as an input.

For the testing of this integration, the set of input files with the business requirements considered various multidimensional roles of the included concepts. This is because the main task of the MDBE system is to finalize the tagging of the MGraph (future MD design) and then to validate this tagging (according to the multidimensional principles). Therefore, two different kinds of test cases are provided.

- First one, with the concept tagged following these principles, and
- Another one disobeying some of these principles. (e.g., for disobeying the multidimensional principles, the input file with business requirements including only level concepts is provided).

After these tests are launched the behavior of the system is analyzed. Since some irregularities are noticed the method for producing the MGraph is examined. After the meeting and examining the elements of the MGraph that needs to be populated the problems are identified. Actually, the elements that are needed to be populated stayed empty. Then the *produceOutputGraph* method is revised. After the revision and re-launching of the tests, system represented the expected behavior and then it was concluded that the integration with the MDBE system was successful.

Afterwards, ***the integration with the Operation Identification stage*** is conducted. Equally, in this case the problem of complexity of ETLGraph structure and missing the comparing methods prevented the JUnit testing for the methods that creates these structures. Therefore, the similar testing technique is introduced here. Only difference is that the one of the inputs for the Operation Identification stage is tagged and validated MGraph which is the product of the integrated MDBE system.

After the tests are launched the behavior of the Operation Identification part was analyzed. At the beginning some exceptions are noticed. After the examining the ETLGraph structure it is identified that some of the names inside the graph do not correspond to the once required from the Operation Identification stage. The main reason for these irregularities comes from the fact that Operation Identification stage was still under the development process and thus some of the requirements changed through the time, but are not updated inside the method for building the initial ETLGraph structure in Requirement Validation stage. After all the differences are resolved retesting showed that the integration with the Operation Identification stage was finally successful.

4.4.5. System testing

After the whole system is built and all individual modules, that I developed, are tested as well as the integration with the MDBE and Operation Identification modules, the testing of the system as a whole is required to ensure the correctness of the system and its overall stability.

For this purpose the set of input test cases are prepared. This set contains the tests which represent all functionalities that GEM offers. Both correct and incorrect inputs should be provided.

Some specific test cases are:

- Concept of the business requirements cannot be identified in the ontology
- Different concepts' mappings that are already explained in section 3.1.2.1.,
- Input XML with the source mappings of the associations produced in order for system to identify multiple paths between the same tagged concepts.
- Tagged concept without the path to any other tagged concept.

After the prepared test cases are launched, the system is analyzed through different stages and it has been noticed that the system works properly, gives the appropriate output designs and appropriately reacts if the inputs with the errors are provided.

5. Cost of the project

For conducting this project inside the real business world the economic study of the necessary resources has been done. Since this project included the theoretical part, i.e., research part, then this part is also included in the evaluation of the project costs.
Therefore, the five different kinds of resources identified as necessary for this project are:

- Research resources
- Hardware resources
- Software resources
- Human resources and
- Office resources

In the next section the time needed for the realization of the project is estimated and then compared to the time actually spent on this project. Afterwards, according to this time, each of the above resources is first explained with its specificities for this project and then its cost is evaluated.

For the estimation of the costs it is considered that the researchers/developers works Monday-Friday, 8 hours per day. At the end, the final amount is calculated and presented.

5.1. Project Length

At this point the time estimated prior the beginning of the project is discussed. Afterwards this time is compared to the time that was actually needed for finalization of this project. When estimating time two different stages of the work is considered:

- Time necessary for the theoretical part of the project
- Time necessary for the technological part of the project

First, the time needed for the solid research process and for making a wide picture of the current state of the art should be considered. Since the researcher inside this project is not already experienced working in this specific field of research the time for making the clear vision about the field has to be included. Therefore, considering all the obstacles that can arise during the research the estimated period for this part of the project is **240 working hours (~1.5 month)**.
Research time can be divided into two parts:

- making the appropriate state of the art (120 working hours), and
- research work on the GEM system (120 working hours)

After the necessary research is conducted the time needed for the development of the project should be estimated. Since this project includes Scrum process for the development method, three different phases of the project should be considered (Figure 10).

- ***Pre-game phase***. This phase includes the description of the whole system and production of the appropriate system architecture. Part of this system is already included in the research part where the system is actually studied in details and afterwards described with the appropriate architecture. Therefore, the time needed for this phase is lower. The only part of this phase that is not included the research part is the production of the initial set of functional and non-functional requirements that need to be fulfilled during the development process. The time needed for this part of the project is estimated to **40 working hours (~1 week)**.

- ***Development phase.*** This phase includes the entire development process of the system. As already stated six different iterations are identified in the process of the development. However, prior the beginning of the development and since the project includes some specific technologies (OWL/JENA), appropriate time should be taken for the developers to study these specific technologies and it can be estimated to **40 working hours (~1 week)**. After the technologies are studied in details the rest of the development process can begin. The estimated time needed for each iteration is given in table 1.

-

	Length (in working hours)	weeks (approx.)
1st sprint	40	1
2nd sprint	80	2
3rd sprint	100	2.5
4th sprint	60	1.5
5th sprint	65	1.5
6th sprint	30	0.5
Total	**375**	**~9**

Table 1: Estimated time for the development phase

- ***Post-game phase.*** This phase included the testing of the system and the time necessary is estimated to **40 working hours (~1 week).** The time estimated for each phase of the testing process (section 4.4) is presented in table 2.

	Length (in working hrs.)	days (approx.)
Modular (Unit) testing	20	2.5
Integration testing	15	2
System testing	5	1
Total	**40**	**~5.5**

Table 2: Estimated time for the testing phase

When the system is finally developed the process of writing the documentation follows. Since the first theoretical part of the project is already documented during the research process this

documentation part includes only the detailed documentation of the technological part of the system. Therefore, time needed for this part is estimated to **120 working hours (~3 weeks).**
After the project is finalized, the total time that actually was needed for its realization is calculated. The time that was really necessary for finalization of the project was higher than the estimated one. These differences are represented in Table 3 and graphically in figure 27.

Phases		***Estimated (hrs.)***	***Real (hrs.)***	***Difference (hrs.)***
Research phase	State of the art	120	100	-20
	GEM	120	200	80
Development phase (Scrum)	Pre-game	40	40	0
	New technologies	40	60	20
	Development	375	450	75
	Post-game (testing)	40	30	-10
Documentation		120	150	30
Total (hrs.)		**855**	**1030**	**175**

Table 3: Difference between estimated and real time spent during the project

It can be noticed that more time was spent during the ***research phase*** studying the new GEM approach, during the ***development phase*** and during the introduction to the ***new technologies***.

- Additional time during the research phase can be justified with the fact that GEM represents new approach which is still under the research and which was therefore the subject to the various changes.

- The time additionally spent during the development phase can be justified by the fact that the development process included dealing with the OWL ontologies that I was not previously familiar with. Additionally, more time was spent during the integrations inside the development phase because:
    - the MDBE system was originally developed for SQL inputs and additional effort and time was spent for adapting MDBE to the new inputs, and because
    - the Operation Identification stage was developed inside the separate project. This project was going in parallel, and thus many things were not precisely defined at the beginning, so the constant communication with Daniel Gonzalez and adaptations to its changes were necessary.

- Finally more time was required for the detail introduction to the JENA library (new technology) since this library is not well documented, but the appropriate forum has to be searched for the useful information.

On the other side, it can be noticed that some phases required less time than estimated.

- The time needed for the State of the art was less than expected, since the wide research of the multidimensional design was already conducted inside the research group.
- Less time was also needed for the testing process since the TPC-H benchmark was used and the test cases were easily translated from the already existing relational schema and queries.

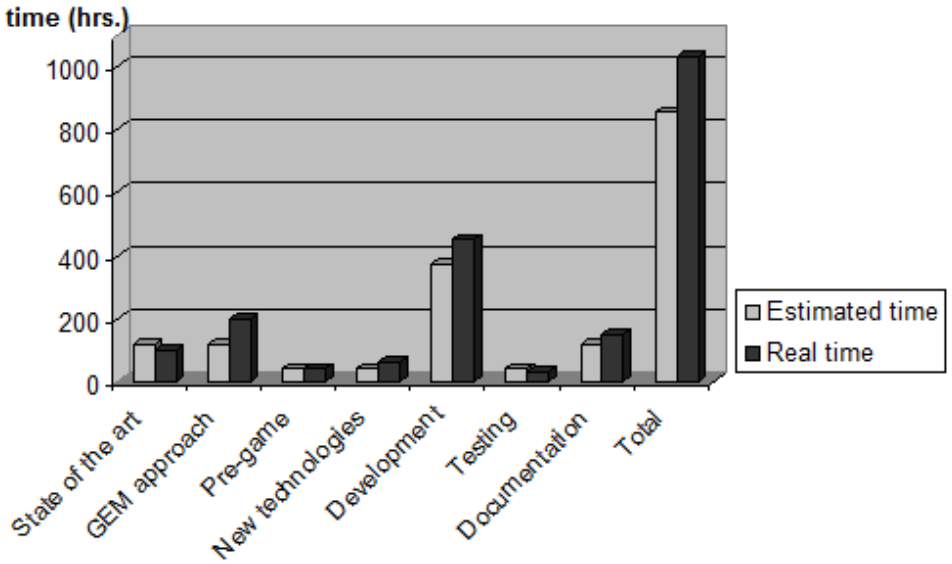


Figure 27: Graphical representation of the time deviations

5.2. Research resources

This estimation considers the research group (1 member) and 1.5 month of the research part.
The resources needed for the realization of the research part of the project are listed in table 4 with their estimated costs.

Resources	Explanation	Total estimated cost
Access to the online collections with the research material	Access to the most prominent collections IEEE, Google scholar, DBLP etc. (~300€/month * 1.5 month)	450€
	Total	450€

Table 4: Costs of the research process

5.3. Hardware resources

This section covers the hardware resources (Computer with the supporting hardware, telecommunication infrastructure etc.) needed during the whole project.

The hardware resources with specificities related to this project and their estimated costs are presented in table 5.

Resources	Explanation	Total estimated cost
Desktop computers	Desktop computer with high performances that would support necessary software needed. Intel® Core™ i5-2400 3.1GHz MSI ATI RADEON R6850 1GB 4GB DDR3-1333 KINGSTON WD 1TB SATAII (~500€ per computer)	500€
Monitor 20”	Quality LCD monitors (~120€ per monitor)	120€
Networking infrastructure	Routers and cable material (~100 €)	100€
Other equipment	Keyboards, mice, etc (~50 €)	50€
Permanent internet access	ADSL internet access (~25€/month * 6 months)	140€
	Total	910€

Table 5: Cost of the hardware resources

5.4. Software resources

This section covers the software resources (operating systems, IDE tools, modeling tools etc.) needed during the whole project.

The software resources with specificities related to this project and their estimated costs are presented in table 6.

Resources	Explanation	Total estimated cost
Operating System	Windows 7 Professional- (~400€ per computer)	400€
IDE tool	NetBeans 7.0 (0€)	0€
UML Design tool	Visual Paradign for UML Modeler Edition (~70 € per computer)..	70€
Smart text editor	Editors for XML and OWL files. Nodepad++ , (0 €)	0€
	Total	470€

Table 6: Cost of the software resources

5.5. Human resources

This section covers the human resources needed during the whole project, i.e., computer science and IT specialists.
The human resources with specificities related to this project and their estimated costs are presented in table 7.

Specialist	Explanation	Total estimated cost
Computer scientist	The computer scientist included in the research process. (~20€/hour per person * 240 hrs.)	4800€
Software Designer	The specialist responsible for the processes of the software design (~25€/hour per person * 80 hrs.). The starting part of every development iteration included the software design process. (~15 hrs. * 6 iterations= 80hrs.)	2000€
Coder	The Java programming specialist for the process of implementation (~20€/hour per person * 180 hrs.) The second part of every development iteration included the software coding process. (~30 hrs. * 6 iterations = 180 hrs.)	3600€
Tester	The testing specialist for the post-game phase (testing process). (~15€ * 40 hrs.)	600€
	Total	11000€

Table 7: Cost of the human resources

5.6. Office resources

This section covers the office resources needed during the whole project. The office resources with specificities related to this project and their estimated costs are presented in table 8.

Finally the total estimated cost of the project like this one is given in table 9, regarding the different kinds of the resources discussed at the beginning. However, since the measured time for the project considered academic environment and not high-experienced developer the time needed for realization of the project like this one in the business environment can be less than one measure here.

Resources	Explanation	Total estimated cost
Properly furnished offices where the research/development group would work	This requires renting the medium-size office (~600€/month * 6 months) and furnishing it with the basic equipment (computer desk, ergonomic chair, plain chairs *4 , desk lamp = ~400€)	4000€
	Total	4000€

Table 8: Cost of the office resources

Resources	Total estimated cost
Research resources	450€
Hardware resources	910€
Software resources	470€
Human resources	11000€
Office resources	4000€
Total	16830€

Table 9: Estimated cost of the project

6. Conclusion

This master thesis considered first theoretical part, which included making the overview in the field of the ETL design and later the detailed research of the new GEM approach for automation of both multidimensional and ETL designs considering the business requirements. Afterwards, the thesis also considered the technological part whose main task was to provide the GEM system with the initial two stages, i.e., Requirement Validation and Requirement Completion. Additionally the technological part of my thesis also included the integration of the already implemented stages of the GEM framework with the initial stages.

Therefore, results of my work in this thesis are:

- The state of the art in the field of automating and customization of the ETL design generation, and
- The two implemented modules for the initial stages of the GEM framework.

The main result of this project is actually the integrated GEM system. This system now receives at its input the data source schema in the form of the OWL ontology and the mappings of the corresponding schema concepts to the real source data stores in XML format. Additionally, GEM at its input also receives the set of business requirements in XML format. As the result GEM generates the multidimensional and ETL conceptual designs. The ability of GEM to receive the inputs that do not consider any specific technology (but OWL ontology and XML) is provided after the development of the Requirement Validation and Requirement Completion stages. Additionally, the appropriate graphical interface is developed, for providing inputs, interacting with the user and presenting the output designs.

Prior the beginning of my work in this project it was necessary for me to learn more about the various principles and technologies that are basis of this project. Therefore, even though I was already familiar with the field of data warehousing, OLAP and multidimensionality, the more detailed study of these areas was mandatory. Also an introduction to the ontology, the OWL and the way of transforming the data stores into the OWL ontology was necessary, along with the delving into details of the OWL language and Java Library for parsing OWL ontologies (JENA), at the beginning of the development phase.

During this project, I have expanded my knowledge in the fields of data warehousing and decision support and I also gained many useful skills. First, the more systematic research work, which is mainly the result of the theoretical part of my thesis. Another one is the familiarity with the techniques used for the conduction of the systematic literature reviews. Furthermore, since this thesis covers the process of integration with the previously implemented modules, the constant need for the communication with other developers made me more matured in the sense to be able

to patiently listen to the other members and to made some valuable conclusions according to given explanations. Considering the technological part, I have become familiar with the Agile software development methods, especially Scrum, which I adapted to the needs of the development phase of this project.

6.1. Future work

The complete GEM framework is fully open for the future upgrades.

There are possibilities for the improvement of the part responsible for the interaction with the users, especially by automating the parts where the system generates suggestions and offers them to the user. These suggestions along with the requests for the user feedback are, as already stated, present at the various points in the framework and thus lower the overall automation of the framework.

Another part that can be considered for the future work is the final stage of the GEM framework, i.e., Conciliation stage, which is briefly explained in section 1.1., but not yet implemented. This stage considers that the GEM has run previous stages for several business requirements, and then it takes the results (designs) obtained for each business requirement and conciliates those results into the single MD and ETL design.

Since it is expected that the GEM could be potentially used for helping the designers during the production of the real and quality decision making systems, another opportunity for the future enhancements arose. This includes associating the GEM framework with some available ETL design tool. One of the most dominant ETL design tool that is also open source, is the Pentaho Kettle tool. Besides being open source, Kettle also offers the Java API for dynamically producing the design of the ETL process inside the Java applications. The main idea is to upgrade GEM with the possibility to translate the ETL process design that is semi-automatically generated as its output into the format that Kettle needs at its input.

7. References

[1] Simitsis, A., Skoutas, D., & Castellanos, M. (2010). Representation of conceptual ETL designs in natural language using Semantic Web technology. Data Knowledge Engineering, 69(1), 96-115.

[2] Skoutas, D. & Simitsis, A. (2006). Designing ETL processes using semantic web technologies. In Proceedings of the 9th ACM International Workshop on Data Warehousing and OLAP (DOLAP), 67-74.

[3] Panos Vassiliadis, Zografoula Vagena, Spiros Skiadopoulos, Nikos Karayannidis, Timos Sellis (2001), Arktos: towards the modeling, design, control and execution of ETL processes, Information Systems 26 (2001) 537–561

[4] Panos Vassiliadis, Alkis Simitsis, Panos Georgantas, and Manolis Terrovitis, A Framework for the Design of ETL Scenarios, J. Eder and M. Missikoff (Eds.): Information Systems (2005)

[5] Skoutas, D., Simitsis, A.: Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data. IJSWIS pp. 1-24 (2007)

[6] Lenz, H., Shoshani, A.: Summarizability in OLAP and Statistical Data Bases. In: SSDBM. pp. 132-143 (1997)

[7] Romero, O., Abelló, A.: Automatic Validation of Requirements to Support Multidimensional Design. Data Knowl. Eng. 69(9), 917-942 (2010)

[8] Husemann, B., Lechtenb□orger, J., Vossen, G.: Conceptual Data Warehouse Modeling. In: DMDW. pp. 1-11 (2000)

[9] Lechtenborger, J., Vossen, G.: Multidimensional Normal Forms for DataWarehouse Design. Information Systems pp. 415-434 (2003)

[10] Mazon, J., Lechtenborger, J., Trujillo, J.: A Survey on Summarizability Issues in Multidimensional Modeling. DKE pp. 1452-1469 (2009)

[11] GEM DaWaK'11: Oscar Romero, Alkis Simitsis, Alberto Abelló. GEM: Requirement- driven Generation of ETL and Multidimensional Conceptual Designs. Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK'11). To appear

[12] Oscar Romero. Automating the Design of Data Warehouses. PhD Thesis, Universitat Politècnica de Catalunya, Barcelona, Spain. http://www.tdx.cat/handle/10803/6670

[13] Oscar Romero, Alberto Abelló. Multidimensional Design Methods for Data Warehousing. In *Integrations of Data Warehousing, Data Mining and Database Technologies: Innovative Approaches*. David Taniar and Li Chen Eds. IGI Global: 2011, pp 78-105

[14] Panos Vassiliadis. A Survey of Extract-Transform-Load Technology. In *Integrations of Data Warehousing, Data Mining and Database Technologies: Innovative Approaches*. David Taniar and Li Chen Eds. IGI Global: 2011, pp 171-199

[15] Beck, Kent; et al. (2001). "Manifesto for Agile Software Development". Agile Alliance. Retrieved 2010-06-14.

[16] Shine Technologies, “Agile Methodologies Survey Results” http://www.shinetech.com/download/attachments/98/ ShineTechAgileSurvey2003-01-17.pdf, December 2007.

[17] "Survey Says: Agile Works in Practice", *Dr. Dobb's Journal*, Vol. 31, No. 9. (2006), 62-64

[18] Pekka Abrahamsson, Outi Salo, Jussi Rankainen & Juhani Warsta: Agile software development methods - Review and analysis, VTT Electronics, 2002.

[19] M. Golfarelli and S. Rizzi. *Data Warehouse Design. Modern Principles and Methodologies.* McGraw-Hill, 2009.Cd

[20] Jena javadoc - http://jena.sourceforge.net/javadoc/index.html

[21] Jena – A Semantic Web Framework for Java - http://jena.sourceforge.net/

[22] The current version of TPC-H 2.14.0 - http://www.tpc.org/tpch/spec/tpch2.14.0.pdf

[23] OWL – Web Ontology Language Overview - http://www.w3.org/TR/owl-features/

[24] XML (Extensible Markup Language) - http://en.wikipedia.org/wiki/XML

Appendix A Framework demo and user manual

This appendix includes the demo presentation of the GEM framework. Through this demo the main functionalities of GEM is presented. The screenshots representing the graphical interface and interaction of the GEM with the user are also included in this demo.

A.1. Input data

For starting the GEM system a set of the input files needs to be provided. As mentioned through this document the GEM framework expects three different files at its input.

- OWL ontology representing data sources
- XML file containing mappings of the ontology concepts
- XML file containing business requirements

For generating these input files, like in the testing process, the TPC-H benchmark is used.

OWL ontology

According to TPC-H schema, the underling OWL ontology is created. This ontology represents the source data stores and for better comprehension about the knowledge covered in this ontology its diagrammatic representation is provided in Figure 28.

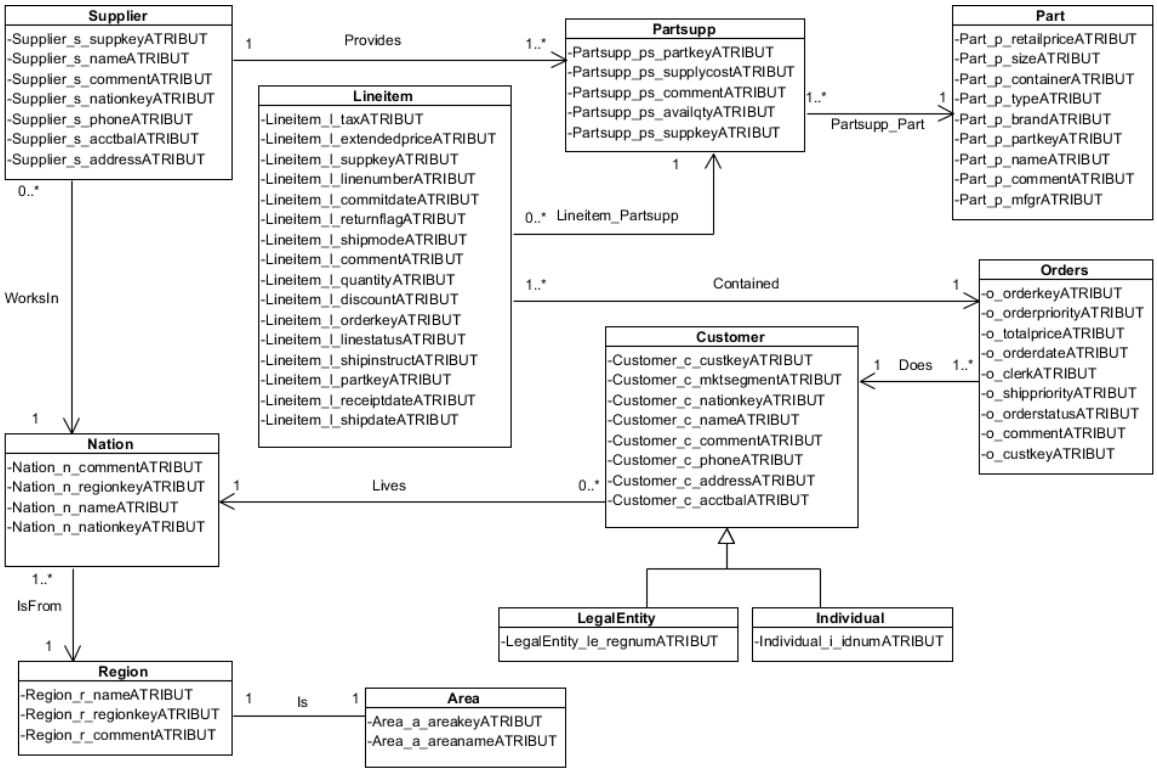


Figure 28: Ontology based on the TPC-H schema

Source mappings

After the OWL ontology is generated, the XML file with the mappings of the ontology concepts is provided.

Considering the format discussed in section 3.1.1.2., the mappings for the ontology classes and their datatype properties are provided.

In table 10 the concepts, for which the mappings are provided, are listed, while in the sequel the complete XML structure with these mappings is provided (Figure 29).

Ontology class	Ontology datatype property (PK – primary key attribute, FK – foreign key attribute)
Nation	Nation_n_nationkeyATRIBUT (PK) Nation_n_regionkeyATRIBUT (FK) Nation_n_nameATRIBUT Nation_n_commentATRIBUT
Region	Region_r_regionkeyATRIBUT (PK) Region_r_nameATRIBUT Region_r_commentATRIBUT
Orders	Orders_o_orderkeyATRIBUT (PK) Orders_o_orderdateATRIBUT Orders_o_custkeyATRIBUT (FK)
Lineitem	Lineitem_l_orderkeyATRIBUT (FK) } (PK) Lineitem_l_linenumberATRIBUT (FK) } Lineitem_l_extendedpriceATRIBUT Lineitem_l_discountATRIBUT
Supplier	Supplier_s_suppkeyATRIBUT (PK) Supplier_s_nationkeyATRIBUT (FK) Supplier_s_nameATRIBUT Supplier_s_phoneATRIBUT Supplier_s_addressATRIBUT
Partsupp	Partsupp_ps_partkeyATRIBUT (FK) } (PK) Partsupp_ps_suppkeyATRIBUT (FK) } Partsupp_ps_supplycostATRIBUT
Part	Part_p_partkeyATRIBUT (PK) Part_p_retailpriceATRIBUT Part_p_typeATRIBUT Part_p_nameATRIBUT
Individual	Individual_i_idnumATRIBUT (PK)
LegalEntity	LegalEntity_le_regnumATRIBUT (PK)

Table 10: Mapped ontology classes and their datatype properties

```
<OntologyMappings>
   <!--THIS IS THE MAPPINGS OF THE CLASS Nation AND ITS DATATYPES -->
   <OntologyMapping sourceKind="relational">
       <Ontology type="concept">
          http://www.owl-ontologies.com/unnamed.owl#Nation
       </Ontology>
       <RefOntology type="property">
          http://www.owl-ontologies.com/unnamed.owl#Nation_n_nationkeyATRIBUT
       </RefOntology>
       <Mapping>
          <Tablename>nation</Tablename>
          <Projections>
                 <Attribute>n_nationkey</Attribute>
          </Projections>
       </Mapping>
   </OntologyMapping>

   <OntologyMapping sourceKind="relational">
       <Ontology type="property">
          http://www.owl-ontologies.com/unnamed.owl#Nation_n_nationkeyATRIBUT
       </Ontology>
       <Mapping>
          <Tablename>nation</Tablename>
          <Projections>
                 <Attribute>n_nationkey</Attribute>
          </Projections>
       </Mapping>
   </OntologyMapping>

   <OntologyMapping sourceKind="relational">
       <Ontology type="property">
          http://www.owl-ontologies.com/unnamed.owl#Nation_n_regionkeyATRIBUT
       </Ontology>
       <RefOntology type="concept">
          http://www.owl-ontologies.com/unnamed.owl#Region
       </RefOntology>
       <Mapping>
          <Tablename>nation</Tablename>
          <Projections>
                 <Attribute>n_regionkey</Attribute>
          </Projections>
       </Mapping>
   </OntologyMapping>

   <OntologyMapping sourceKind="relational">
       <Ontology type="property">
          http://www.owl-ontologies.com/unnamed.owl#Nation_n_commentATRIBUT
       </Ontology>
       <Mapping>
          <Tablename>nation</Tablename>
          <Projections>
                 <Attribute>n_comment</Attribute>
                 <Attribute>n_nationkey</Attribute>
          </Projections>
       </Mapping>
   </OntologyMapping>

   <OntologyMapping sourceKind="relational">
       <Ontology type="property">
          http://www.owl-ontologies.com/unnamed.owl#Nation_n_nameATRIBUT
       </Ontology>
       <Mapping>
          <Tablename>nation</Tablename>
          <Projections>
                 <Attribute>n_name</Attribute>
                 <Attribute>n_nationkey</Attribute>
          </Projections>
       </Mapping>
```

```
</OntologyMapping>

<!--THIS IS THE MAPPINGS OF THE CLASS Region AND ITS DATATYPES -->
<OntologyMapping sourceKind="relational">
    <Ontology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Region
    </Ontology>
    <RefOntology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Region_r_regionkeyATRIBUT
    </RefOntology>
    <Mapping>
       <Tablename>region</Tablename>
       <Projections>
              <Attribute>r_regionkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Region_r_regionkeyATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>region</Tablename>
       <Projections>
              <Attribute>r_regionkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Region_r_nameATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>region</Tablename>
       <Projections>
              <Attribute>r_name</Attribute>
              <Attribute>r_regionkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Region_r_commentATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>region</Tablename>
       <Projections>
              <Attribute>r_comment</Attribute>
              <Attribute>r_regionkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<!--THIS IS THE MAPPINGS OF THE CLASS Orders AND ITS DATATYPES -->
<OntologyMapping sourceKind="relational">
    <Ontology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Orders
    </Ontology>
    <RefOntology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Orders_o_orderkeyATRIBUT
    </RefOntology>
    <Mapping>
       <Tablename>orders</Tablename>
       <Projections>
              <Attribute>o_orderkey</Attribute>
```

```
        </Projections>
      </Mapping>
  </OntologyMapping>

  <OntologyMapping sourceKind="relational">
      <Ontology type="property">
        http://www.owl-ontologies.com/unnamed.owl#Orders_o_custkeyATRIBUT
      </Ontology>
      <RefOntology type="concept">
        http://www.owl-ontologies.com/unnamed.owl#Customer
      </RefOntology>
      <Mapping>
        <Tablename>orders</Tablename>
        <Projections>
              <Attribute>o_custkey</Attribute>
        </Projections>
      </Mapping>
  </OntologyMapping>


  <OntologyMapping sourceKind="relational">
      <Ontology type="property">
        http://www.owl-ontologies.com/unnamed.owl#Orders_o_orderdateATRIBUT
      </Ontology>
      <Mapping>
        <Tablename>orders</Tablename>
        <Projections>
              <Attribute>o_orderdate</Attribute>
              <Attribute>o_orderkey</Attribute>
        </Projections>
      </Mapping>
  </OntologyMapping>

  <!--THIS IS THE MAPPINGS OF THE CLASS Lineitem AND ITS DATATYPES -->
  <OntologyMapping sourceKind="relational">
      <Ontology type="concept">
        http://www.owl-ontologies.com/unnamed.owl#Lineitem
      </Ontology>
      <RefOntology type="property">
        http://www.owl-ontologies.com/unnamed.owl#Lineitem_l_orderkeyATRIBUT
      </RefOntology>
      <Mapping>
        <Tablename>lineitem</Tablename>
        <Projections>
              <Attribute>l_orderkey</Attribute>
        </Projections>
      </Mapping>
  </OntologyMapping>

  <OntologyMapping sourceKind="relational">
      <Ontology type="concept">
        http://www.owl-ontologies.com/unnamed.owl#Lineitem
      </Ontology>
      <RefOntology type="property">
        http://www.owl-
ontologies.com/unnamed.owl#Lineitem_l_linenumberATRIBUT
      </RefOntology>
      <Mapping>
        <Tablename>lineitem</Tablename>
        <Projections>
              <Attribute>l_linenumber</Attribute>
        </Projections>
      </Mapping>
  </OntologyMapping>

  <OntologyMapping sourceKind="relational">
      <Ontology type="property">
        http://www.owl-ontologies.com/unnamed.owl#Lineitem_l_orderkeyATRIBUT
```

```
    </Ontology>
    <RefOntology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Orders
    </RefOntology>
    <Mapping>
       <Tablename>lineitem</Tablename>
       <Projections>
              <Attribute>l_orderkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Lineitem_l_partkeyATRIBUT
    </Ontology>
    <RefOntology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Partsupp
    </RefOntology>
    <Mapping>
       <Tablename>lineitem</Tablename>
       <Projections>
              <Attribute>l_partkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Lineitem_l_suppkeyATRIBUT
    </Ontology>
    <RefOntology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Partsupp
    </RefOntology>
    <Mapping>
       <Tablename>lineitem</Tablename>
       <Projections>
              <Attribute>l_suppkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
   http://www.owl-ontologies.com/unnamed.owl#Lineitem_l_extendedpriceATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>lineitem</Tablename>
       <Projections>
              <Attribute>l_orderkey</Attribute>
              <Attribute>l_linenumber</Attribute>
              <Attribute>l_extendedprice</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Lineitem_l_discountATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>lineitem</Tablename>
       <Projections>
              <Attribute>l_orderkey</Attribute>
              <Attribute>l_linenumber</Attribute>
              <Attribute>l_discount</Attribute>
       </Projections>
    </Mapping>
```

```
</OntologyMapping>

<!--THIS IS THE MAPPINGS OF THE CLASS Supplier AND ITS DATATYPES -->
<OntologyMapping sourceKind="relational">
    <Ontology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Supplier
    </Ontology>
    <RefOntology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Supplier_s_suppkeyATRIBUT
    </RefOntology>
    <Mapping>
       <Tablename>supplier</Tablename>
       <Projections>
              <Attribute>s_suppkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Supplier_s_nationkeyATRIBUT
    </Ontology>
    <RefOntology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Nation
    </RefOntology>
    <Mapping>
       <Tablename>supplier</Tablename>
       <Projections>
              <Attribute>s_nationkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Supplier_s_suppkeyATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>supplier</Tablename>
       <Projections>
              <Attribute>s_suppkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Supplier_s_nameATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>supplier</Tablename>
       <Projections>
              <Attribute>s_name</Attribute>
              <Attribute>s_suppkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Supplier_s_phoneATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>supplier</Tablename>
       <Projections>
              <Attribute>s_phone</Attribute>
              <Attribute>s_suppkey</Attribute>
       </Projections>
```

```
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Supplier_s_addressATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>supplier</Tablename>
       <Projections>
              <Attribute>s_address</Attribute>
              <Attribute>s_suppkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<!--THIS IS THE MAPPINGS OF THE CLASS Partsupp AND ITS DATATYPES -->
<OntologyMapping sourceKind="relational">
    <Ontology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Partsupp
    </Ontology>
    <RefOntology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Partsupp_ps_partkeyATRIBUT
    </RefOntology>
    <Mapping>
       <Tablename>partsupp</Tablename>
       <Projections>
              <Attribute>ps_partkey</Attribute>
              <Attribute>ps_suppkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Partsupp
    </Ontology>
    <RefOntology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Partsupp_ps_suppkeyATRIBUT
    </RefOntology>
    <Mapping>
       <Tablename>partsupp</Tablename>
       <Projections>
              <Attribute>ps_suppkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Partsupp_ps_partkeyATRIBUT
    </Ontology>
    <RefOntology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Part
    </RefOntology>
    <Mapping>
       <Tablename>partsupp</Tablename>
       <Projections>
              <Attribute>ps_partkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Partsupp_ps_suppkeyATRIBUT
    </Ontology>
    <RefOntology type="concept">
```

```
        http://www.owl-ontologies.com/unnamed.owl#Supplier
    </RefOntology>
    <Mapping>
       <Tablename>partsupp</Tablename>
       <Projections>
              <Attribute>ps_suppkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>
<OntologyMapping sourceKind="relational">
    <Ontology type="property">
    http://www.owl-ontologies.com/unnamed.owl#Partsupp_ps_supplycostATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>partsupp</Tablename>
       <Projections>
              <Attribute>ps_supplycost</Attribute>
              <Attribute>ps_suppkey</Attribute>
              <Attribute>ps_partkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<!--THIS IS THE MAPPINGS OF THE CLASS Part AND ITS DATATYPES -->
<OntologyMapping sourceKind="relational">
    <Ontology type="concept">
       http://www.owl-ontologies.com/unnamed.owl#Part
    </Ontology>
    <RefOntology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Part_p_partkeyATRIBUT
    </RefOntology>
    <Mapping>
       <Tablename>part</Tablename>
       <Projections>
              <Attribute>p_partkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Part_p_sizeATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>part</Tablename>
       <Projections>
              <Attribute>p_size</Attribute>
              <Attribute>p_partkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Part_p_typeATRIBUT
    </Ontology>
    <Mapping>
       <Tablename>part</Tablename>
       <Projections>
              <Attribute>p_type</Attribute>
              <Attribute>p_partkey</Attribute>
       </Projections>
    </Mapping>
</OntologyMapping>

<OntologyMapping sourceKind="relational">
    <Ontology type="property">
       http://www.owl-ontologies.com/unnamed.owl#Part_p_partkeyATRIBUT
```

```
      </Ontology>
      <Mapping>
        <Tablename>part</Tablename>
        <Projections>
              <Attribute>p_partkey</Attribute>
        </Projections>
      </Mapping>
  </OntologyMapping>

  <OntologyMapping sourceKind="relational">
      <Ontology type="property">
        http://www.owl-ontologies.com/unnamed.owl#Part_p_retailpriceATRIBUT
      </Ontology>
      <Mapping>
        <Tablename>part</Tablename>
        <Projections>
              <Attribute>p_retailprice</Attribute>
        </Projections>
      </Mapping>
  </OntologyMapping>

  <!--THIS IS THE MAPPINGS OF THE CLASS LegalEntity AND ITS DATATYPES -->
  <OntologyMapping sourceKind="relational">
      <Ontology type="concept">
        http://www.owl-ontologies.com/unnamed.owl#LegalEntity
      </Ontology>
      <RefOntology type="property">
        http://www.owl-
ontologies.com/unnamed.owl#LegalEntity_le_regnumATRIBUT
      </RefOntology>
      <Mapping>
        <Tablename>legal_entity</Tablename>
        <Projections>
              <Attribute>le_regnum</Attribute>
        </Projections>
      </Mapping>
  </OntologyMapping>

  <!--THIS IS THE MAPPINGS OF THE CLASS Individual AND ITS DATATYPES -->
  <OntologyMapping sourceKind="relational">
      <Ontology type="concept">
        http://www.owl-ontologies.com/unnamed.owl#Individual
      </Ontology>
      <RefOntology type="property">
        http://www.owl-ontologies.com/unnamed.owl#Individual_i_idnumATRIBUT
      </RefOntology>
      <Mapping>
        <Tablename>individual</Tablename>
        <Projections>
              <Attribute>i_idnum</Attribute>
        </Projections>
      </Mapping>
  </OntologyMapping>

  <!--THE MAPPINGS OF THE ASSOCIATIONS -->
  <OntologyMapping sourceKind="relational">
      <Ontology type="property">
        http://www.owl-ontologies.com/unnamed.owl#IsFrom
      </Ontology>
      <Mapping>
        <Tablename>nation</Tablename>
        <Projections>
              <Attribute>n_nationkey</Attribute>
              <Attribute>n_regionkey</Attribute>
        </Projections>
      </Mapping>
  </OntologyMapping>
```

```
   <OntologyMapping sourceKind="relational">
       <Ontology type="property">
          http://www.owl-ontologies.com/unnamed.owl#Lives
       </Ontology>
       <Mapping>
          <Tablename>customer</Tablename>
          <Projections>
                 <Attribute>c_nationkey</Attribute>
                 <Attribute>c_custkey</Attribute>
          </Projections>
       </Mapping>
   </OntologyMapping>

   <OntologyMapping sourceKind="relational">
       <Ontology type="property">
          http://www.owl-ontologies.com/unnamed.owl#Does
       </Ontology>
       <Mapping>
          <Tablename>orders</Tablename>
          <Projections>
                 <Attribute>o_custkey</Attribute>
                 <Attribute>o_orderkey</Attribute>
          </Projections>
       </Mapping>
   </OntologyMapping>

   <OntologyMapping sourceKind="relational">
       <Ontology type="property">
          http://www.owl-ontologies.com/unnamed.owl#Contained
       </Ontology>
       <Mapping>
          <Tablename>lineitem</Tablename>
          <Projections>
                 <Attribute>l_orderkey</Attribute>
                 <Attribute>l_linenumber</Attribute>
                 <Attribute>l_orderkey</Attribute>
          </Projections>
       </Mapping>
   </OntologyMapping>
</OntologyMappings>
```

Figure 29: Complete XML structure for the source mappings used in the demo

It should be noticed that the mapping for the concept Customer is not provided since this demo should represent how the system reacts in the case that the new derived mapping should be provided.

As it can be seen in figure 29, the mappings of some of following associations (ontology properties) are also provided:

- IsFrom
- Lives
- Does
- Contained

The set of the mappings is intentionally made to be able to represent some GEM functionalities. (e.g., derived mapping of the concept *Customer* from its mapped subclasses – *LegalEntity* and *Individual*)

After the file with the mapping of these concepts is generated, the file containing business requirements is provided. The content of this file is presented in Figure 30.

This XML structure is derived from the QUERY 5 of the TPC-H benchmark and it aims at listing the sums of potential revenues, expressed with function Lineitem_l_extendedpriceATRIBUT* (1-Lineitem_l_discountATRIBUT), for each nation (Nation_n_nameATRIBUT) and according to orders done before the date 18/04/2011, the customer with the name CUSTOMER and the region with the name REGION.

Note that the names in the XML in figure 30 are extended to correspond to the names from the ontology. According to these inputs the system demo execution is depicted in the figures following figure 30.

```
<?xml version="1.0"?>
<!DOCTYPE cube SYSTEM "cube.dtd">
<cube>
 <dimensions>
  <concept id="Nation_n_nameATRIBUT" />
 </dimensions>
 <measures>
  <concept id="revenue">
        <function>
              Lineitem_l_extendedpriceATRIBUT*(1-Lineitem_l_discountATRIBUT)
        </function>
  </concept>
 </measures>
 <slicers>
  <comparison>
        <concept id="Orders_o_orderdateATRIBUT" />
        <operator>&lt;</operator>
        <value>18/04/2011</value>
  </comparison>
  <comparison>
        <concept id="Region_r_nameATRIBUT" />
        <operator>=</operator>
        <value>REGION</value>
  </comparison>
  <comparison>
        <concept id="Customer_c_nameATRIBUT" />
        <operator>=</operator>
        <value>CUSTOMER</value>
  </comparison>
 </slicers>
 <aggregations>
  <aggregation order="1">
        <dimension refID="Nation_n_nameATRIBUT" />
        <measure refID="revenue" />
        <function>SUM</function>
  </aggregation>
 </aggregations>
</cube>
```

Figure 30: Input XML file with the business requirements

A.2. Resulting execution

After the GEM is started the main screen appears. (Figure 31)

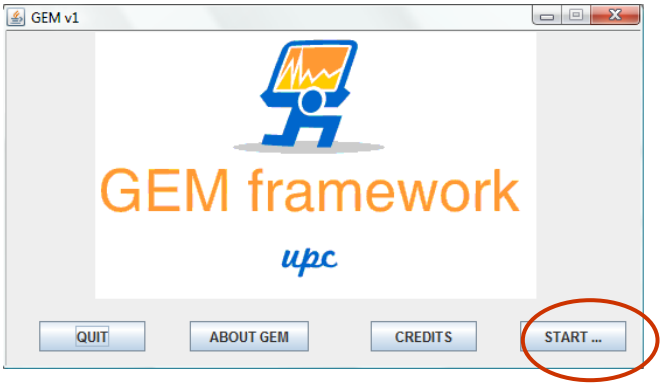


Figure 31: Main screen of the GEM framework

The main screen contains several options:

- START… – to start working with the GEM system
- CREDITS – listing the people credited for the research and the development of the GEM system
- ABOUT GEM – gives a brief introduction to GEM and its possibilities.
- QUIT – to exit from the GEM

Since other options are straightforward the option START… is chosen. After this option is chosen, the following screen appears. (Figure 32)

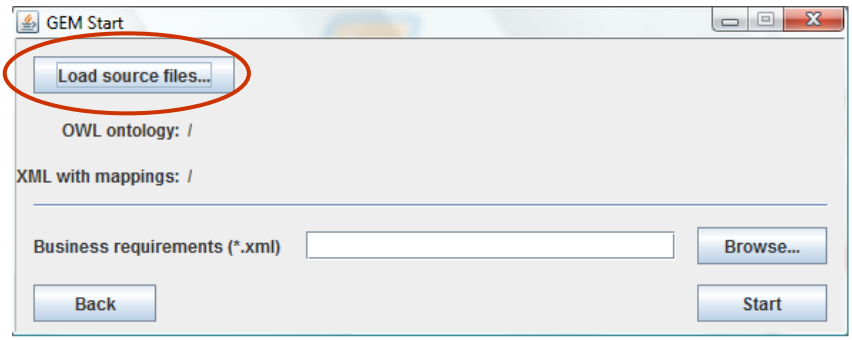


Figure 32: Beginning with GEM

This screen is the starting point for working with the GEM system. Two separated parts can be noticed in this window. Upper one contains a button “Load source files…”. This part of the window is used for loading the OWL ontology and XML file with mapping. After the button is clicked the window for loading the corresponding files appears (Figure 33).

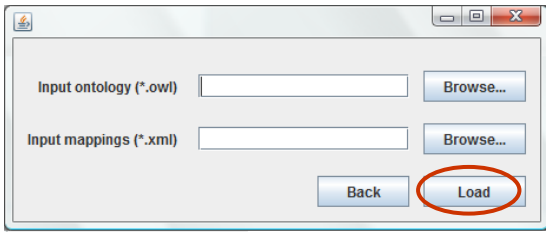


Figure 33: Loading of OWL and mappings

Choosing the “Browse…” options the standard window for file choosing appears and after both files are chosen the “Load“ button is clicked to start loading of the chosen files to the

corresponding structures. Therefore, the system goes back to the previous screen where now the paths of the loaded files are showed. (Figure 34).

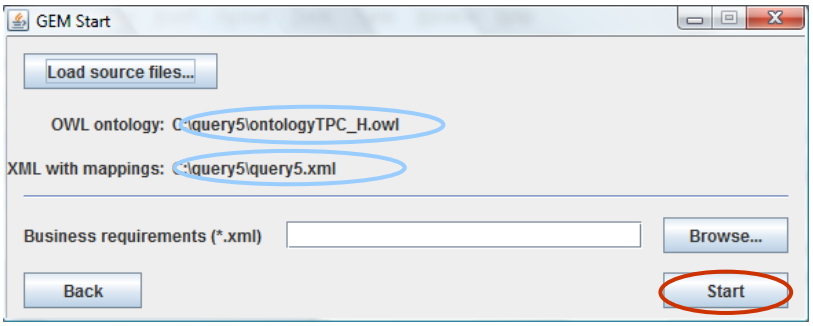


Figure 34: OWL ontology and XML with mappings are loaded

After the first two input files are loaded into the internal structures, now the XML file with the business requirements is expected. After the button "Browse…" is clicked the standard file chooser window appears. When the XML file is chosen, the option "Start" is clicked to start the GEM process. After the button is clicked in the background the chosen XML file is loaded into the corresponding structure and then the control screen of the GEM process appears. (Figure 35).

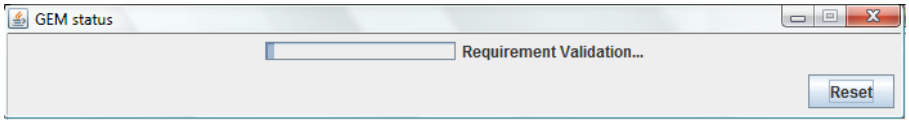


Figure 35: Start of the Requirement Validation stage

During this stage, more specifically during the source mapping process, the system will ask for the user's feedback. This happens since the input business requirement file contains the concept Customer_c_nameATRIBUT. This concept is identified as a datatype property which domain class is Customer. However inside the XML file with the source mappings the concept Customer is not mapped, but its subclasses are (Individual and LegalItem). Therefore, the system produces the corresponding suggestion for the derived mapping and presente it to the user (Figure 36).

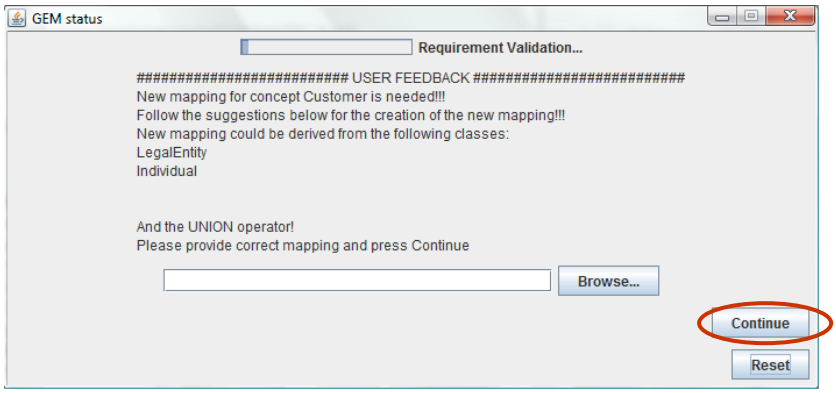


Figure 36: Interaction with the user (derived mapping file is expected)

At this point the user is supposed to create new XML file that will contained the derived mapping according to the suggestions. The corresponding XML file and the derived mapping is presented in Figure 37.

```
<OntologyMappings>

   <OntologyMapping sourceKind="relational">
       <Ontologyntype = concept ">
          http://www.owl-ontologies.com/unnamed.owl#Customer
       </Ontology>
       <RefOntology type="property">
         http://www.owl-ontologies.com/unnamed.owl#Customer_c_custkeyATRIBUT
       </RefOntology>
       <Mapping>
              <Mapping>
                     <Tablename>legal_entity</Tablename>
                     <Projections>
                            <Attribute>le_regnum</Attribute>
                     </Projections>
              </Mapping>
              <SQLOperator>UNION</SQLOperator>
              <Mapping>
                     <Tablename>individual</Tablename>
                     <Projections>
                            <Attribute>i_idnum</Attribute>
                     </Projections>
              </Mapping>
       </Mapping>
   </OntologyMapping>

</OntologyMappings>
```

Figure 37: Feedback XML file with the derived mapping

After the new file is loaded ("Browse…") the option "Continue" is chosen to continue with the mapping process. In the background the new feedback file is loaded and added to the previous structure. The options for loading the feedback file then disappear and the system continues. After the stage of Requirement Validation is finished, the progress bar advances, the stage of the requirement completion starts and the control screen changes (Figure 38).

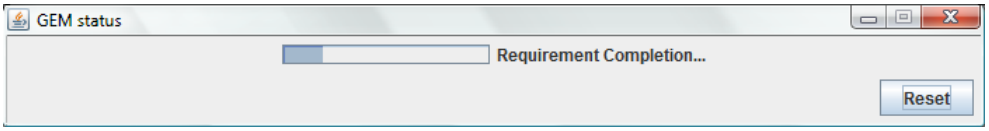


Figure 38: Control screen during the Requirement Completion stage

Afterwards, when the stage of Requirement Completion is finished the following stage of Multidimensional Validation (MDBE) starts (Figure 39).

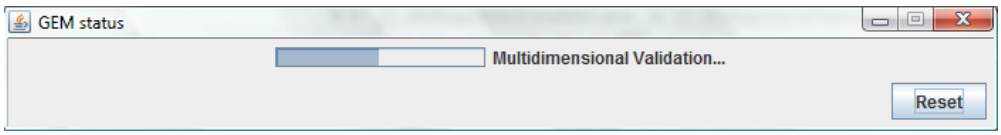


Figure 39: Control screen during the Multidimensional Validation stage

After the stage of Multidimensional Validation, the system starts with the stage of Operation Identification (Figure 40).

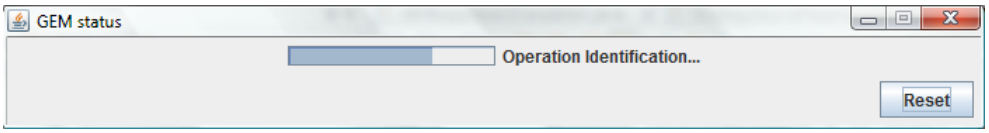


Figure 40: Control screen during the Operation Identification stage

By the end of the Operation Identification stage, the MD and ETL designs are created and thus the options for their showing are given to the user (Figure 41).

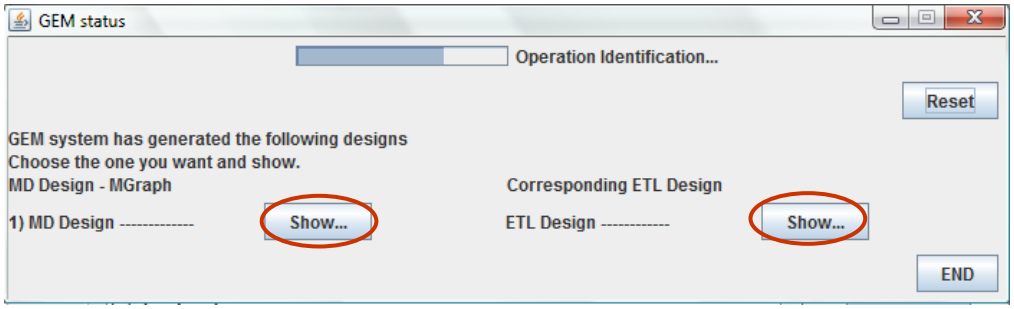


Figure 41: Control screen at the end of Operation Identification stage

Since in this example only one MD design is possible therefore the user is provided with one MD and one corresponding ETL design. These designs can be shown choosing the options "Show…" next to the corresponding design. After choosing the option "Show…" next to the MD design, the window with the graph representing the produced design appears (Figure 42).

Similarly, after choosing the option "Show..." next to the ETL design the window containing the appropriate design appears (Figure 43).

Since ETL design tends to become very complex, the user is provided with two options for searching the details of one operation, represented by the node of the output graph.

The first one is the zoom options (+/-) whit which the user can zoom to the part of the graph he/she is interested in. Another, and maybe more detailed option it to click on the node. After the user clicked on the node the details about this node and corresponding operation is shown (Figure 43).

Figure 42: Generated MD design

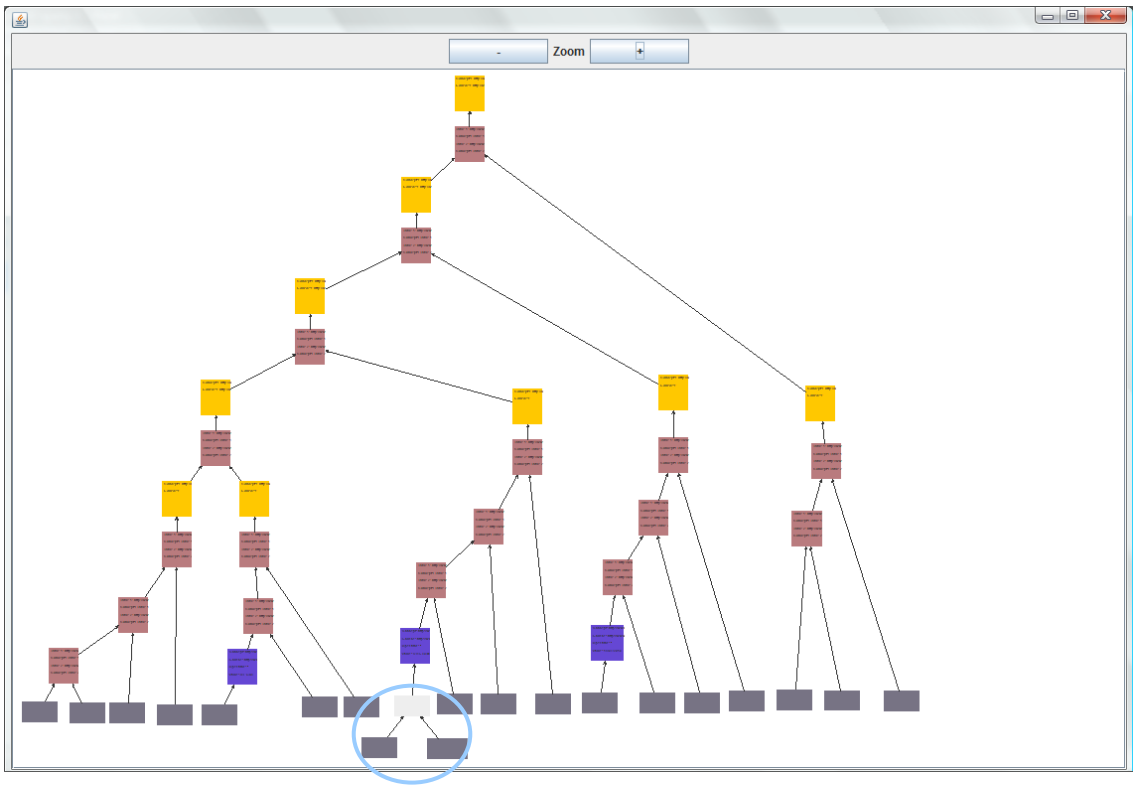


Figure 43: Generated ETL design

In figure 42 the circled subgraph represents the necessary operation for the extraction of the concept Customer. This kind of the structure is the result of the derived mapping provided for the concept Customer during the Requirement Validation stage.

Figure 43: More details about the operation nodes

In figure 43, in the left, the details about the EXTRACTION operation, needed to extract the mapped subclass (Individual) of the concept Customer, is presented. In the right, of the same figure, the details about the UNION operation over the results of the EXTRACTIONs of the mapped subclass concepts (Individual and LegalEntity), is presented.

After the designs are examined the option “Reset” gives chance to go back to the beginning of the GEM system (Figure 34). There the OWL ontology and XML mapping files are already loaded, and the new business requirement file is expected. Otherwise option “Back” returns user to the main screen where he/she can either start GEM again or quit the system.

Appendix B Document Type Definitions for input XML files

```
<!ELEMENT cube (dimensions,measures,slicers?,aggregations?)>
<!ELEMENT dimensions (concept+)>
<!ELEMENT concept (function?,role?,nfr*)*>
<!ELEMENT function (#PCDATA)>
<!ELEMENT role (#PCDATA)>
<!ELEMENT nfr (#PCDATA)>
<!ELEMENT measures (concept*)>
<!ELEMENT slicers (comparison+)>
<!ELEMENT comparison (concept,function?,operator,value)>
<!ELEMENT operator (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT aggregations (aggregation*)>
<!ELEMENT aggregation (dimension,measure,function?)>
<!ELEMENT dimension EMPTY>
<!ELEMENT measure EMPTY>

<!ATTLIST concept id CDATA #REQUIRED>
<!ATTLIST concept alias CDATA #IMPLIED>
<!ATTLIST nfr kind (freshness|precision) #REQUIRED>
<!ATTLIST nfr format CDATA #IMPLIED>
<!ATTLIST nfr value CDATA #IMPLIED>
<!ATTLIST aggregation order CDATA #IMPLIED>
<!ATTLIST dimension refID CDATA #REQUIRED>
<!ATTLIST dimension refAlias CDATA #IMPLIED>
<!ATTLIST measure refID CDATA #REQUIRED>
<!ATTLIST measure refAlias CDATA #IMPLIED>
```

Figure 44: DTD for the XML file with the business requirements

```
<!ELEMENT OntologyMappings (OntologyMapping+)>
<!ELEMENT OntologyMapping (Ontology,RefOntology?,Mapping)>
<!ELEMENT SQLOperator (#PCDATA)>
<!ELEMENT Ontology (#PCDATA)>
<!ELEMENT RefOntology (#PCDATA)>
<!ELEMENT Mapping
(((Mapping,SQLOperator)*,Mapping)|(Tablename,Projections,Selections?))>
<!ELEMENT Tablename (#PCDATA)>
<!ELEMENT Projections (Attribute+)>
<!ELEMENT Attribute (#PCDATA)>
<!ELEMENT Selections (Selection+)>
<!ELEMENT Selection (Column,Operator,Constant)>
<!ELEMENT Column (#PCDATA)>
<!ELEMENT Operator (#PCDATA)>
<!ELEMENT Constant (#PCDATA)>


<!ATTLIST OntologyMapping sourceKind CDATA #REQUIRED>
<!ATTLIST Ontology type (concept|property) #REQUIRED>
<!ATTLIST RefOntology type (concept|property) #REQUIRED>
```

Figure 45: DTD for the XML file with the source mappings

Appendix C Glossary

Agile Software Development	Group of software development methods based on iterative and incremental development
Business Intelligence (BI)	Refers to computer-based techniques used in identifying, extracting, and analyzing business data, to support better business decision-making.
Data Warehousing (DW)	Aims at combining data from multiple and usually varied sources into one comprehensive and easily manipulated database with the main goal to help business in decision making process
Descriptors	In multidimensionality, represent the means for designing the specific subset of values of a certain level
Dimensions	Multidimensional concepts that represent the perspective from which the data are analyzed
Document Type Definitions (DTD)	Formal syntax that declares precisely which elements and references may appear where in the markup document (XML), and what the elements' contents and attributes are.
Extract-Transform-Load (ETL)	A process in data warehousing systems that involves Extracting data from outside sources, Transforming it to fit operational needs and Loading it into the target data warehouse.
Extensible Markup Language (*XML*)	A markup language for documents that are supposed to contain structured information and that are supposed to be exchanged among different systems and platforms.
GEM framework	The framework for semi-automatic Generation of ETL and Multidimensional designs according to the available data sources and previously gathered business requirements.
Integrated development environment (IDE)	A software application that provides environment with the different features that aims to suit almost all programmers needs during the software development process
MDBE system	The system developed by professor Oscar Romero aims at producing multidimensional design based on the underlying data sources and guided with the previously gathered user requirements.
Measures	Multidimensional concepts that represent the data of interest for the analysis process.
Multidimensionality (MD)	Represents the paradigm for analyzing the desired data from the multidimensional point of view.

(Online Analytical Processing) OLAP	On-Line Analytical Processing (OLAP) is a category of software technology that enables analysts, managers and executives to gain insight into the companies data based on multidimensionality
Ontology	A formal representation of certain knowledge as a set of *concepts* within a domain, and the *relationships* between those concepts.
Ontology reasoner	A piece of software able to infer logical consequences from a formally defined set of rules among the domain concepts (ontology).
OWL Web Ontology Language	The language created for defining ontology documents aims to support machine interpretability of Web content.
Plan-driven software development	Formal specific approach for developing a software product strictly following the defined phases and respecting defined roles.
Semantic Web	Approach aims at enabling machines to understand the semantics, or meaning, of information on the World Wide Web.
***Uniform Resource Identifier* (*URI*)**	A string of characters used to identify a name or a resource on the Internet. Ontology URI – unified identifier of the ontology elements (classes and properties)
World Wide Web Consortium (W3C)	The intentional organization with the main purpose of developing standards for the World Wide Web.