Federal University of Rio Grande do Norte
Center of Exact and Earth Sciences
Department of Informatics and Applied Mathematics
Computer Systems Graduation Program

Model-Driven Requirements Engineering Process aided by Ontologies and Natural Controlled Languages

Raphael Mendes de Oliveira Cóbe

Natal
June, 2009

Federal University of Rio Grande do Norte
Center of Exact and Earth Sciences
Department of Informatics and Applied Mathematics
Computer Systems Graduation Program

Model-Driven Requirements Engineering Process aided by Ontologies and Natural Controlled Languages

A Thesis Presented to the Computer Systems Graduation Program at the Department of Informatics and Applied Mathematics department of Federal University of Rio Grande do Norte In Partial Fulfillment of the Requirements for the Degree MASTER OF SCIENCE in Computer Systems.

Raphael Mendes de Oliveira Cóbe

Dr. Paulo Pires
Teacher Advisor

Natal, June, 2009

Abstract

Researches in Requirements Engineering have been growing in the latest few years. Researchers are concerned with a set of open issues such as: communication between several user profiles involved in software engineering; scope definition; volatility and traceability issues. To cope with these issues a set of works are concentrated in (i) defining processes to collect client's specifications in order to solve scope issues; (ii) defining models to represent requirements to address communication and traceability issues; and (iii) working on mechanisms and processes to be applied to requirements modeling in order to facilitate requirements evolution and maintenance, addressing volatility and traceability issues. We propose an iterative Model-Driven process to solve these issues, based on a double layered CIM to communicate requirements related knowledge to a wider amount of stakeholders. We also present a tool to help requirements engineer through the RE process. Finally we present a case study to illustrate the process and tool's benefits and usage.

Research Area: Requirements and Software Engineering
Key words: Model-Driven development; Requirements Engineering process; Natural Controlled Languages; Ontologies; ATL.

Resumo

Pesquisas em Engenharia de Requisitos tem crescido ao longo dos últimos anos. Pesquisadores estão preocupados com uma série de problemas em aberto como: comunicação entre diversos perfís envolvidos na engenharia de requisito; problemas na definição de escopo; problemas de volatilidade e ratreabilidade de requisitos. Para lidar com este conjunto de problemas em aberto, um conjunto de trabalhos estão concentrados em (i) definir processos para coletar especificações de clientes para lidar com prolemas de escopo; (ii) definir modelos para representar requisitos para lidar com problemas de comunicação e rastreabilidade; e (iii) definição de mecanismos e processos para serem aplicados a modelagem de requisitos para facilitar a evolução e manutenção de requisitos, lidando com problemas de volatilidade e rastreabilidade em requisitos. Neste trabalho é proposto um processo dirigido por modelo para solucionar estes problemas em aberto. Este processo é baseado na idéia de um CIM dividido em duas camadas de forma que se consiga difundir o conhecimento relacionado ao requisitos para um número maior de stakeholders. Durante este trabalho também foi desenvolvida uma ferramenta que tem como objetivo auxiliar a execução de tal processo. Por fim apresentamos um estudo de caso para ilustrar os benefícios do uso da ferramenta e do processo.

Area de Concentração: Engenharia de Software e Requisitos

Palavras-Chave: Desenvolvimento Dirigido por Modelos; Processos para Engenharia de Requisitos; Linguagens Naturais Controladas; Ontologias; ATL.

Contents

1 Introduction **1**

1.1 Problems in RE Processes 2

1.2 Main Goals 6

1.3 Proposed Strategy 7

2 Background **9**

2.1 Requirements Engineering 9

2.1.1 Requirements Elicitation and Analysis 10

2.1.2 Requirements Specification 11

2.1.3 Requirements Validation 12

2.2 Model-Driven Engineering 12

2.2.1 Models, Metamodels and Meta-Metamodels 14

2.2.2 Model Transformation 16

2.3 Ontologies 19

2.3.1 RDF(S) and the Ontology Web Language - OWL 22

2.4 Natural Controlled Language - NCL 24

2.4.1 *Attempto Controlled English* - ACE 26

3 Proposed Approach **31**

3.1 Proposed Models and Views 31

3.1.1 Client's View 32
3.1.2 Development Team's View 34
3.1.3 PIM 36
3.2 Proposed process 37
3.2.1 Detailed Process 38
3.3 Tool Support 44
3.3.1 Proposed tool architecture 47

4 Case Study **60**
4.1 The Goal/Question/Metrics plan 61
4.2 The Case Study Scenario 72
4.3 The Process Execution 73
4.3.1 Inconsistencies and Taxonomic Classification 77
4.4 The Case Study Evaluation 89

5 Related Works **102**

6 Final Remarks **105**
6.1 General Comments and Case Study Evaluation 105
6.2 Difficulties 106
6.3 Future Works 107

A Interviews **120**

List of Figures

2.1 Requirements Engineering Process 10

2.2 Several PSMs automatic obtained from a single PIM 14

2.3 MDA's 4 layered approach 15

2.4 Mapping between PIM and PSM in MDA's context [Fra03] 17

2.5 Mapping between CIM, PIM and PSM in MDE's context 17

2.6 ACE Parser Engine WebClient 29

2.7 ACEView Protégé Plugin 30

3.1 4 Layered CIM and partial PIM 32

3.2 The proposed process (UML 2.0) 39

3.3 Message given by the Reasoner while trying to reason at an inconsistent ontology 42

3.4 NCL editor showing conflicting sentences 42

3.5 Proposed Process with specification of Used Tools 45

3.6 The New ACEView 47

3.7 RE Tool's Architecture (UML 2.0) 48

3.8 Main Classes of the ACEView Domain 49

3.9 ACEView new Views 52

3.10 ACEView's ACESnippetEditor 53

3.11 ACEView's ACEIndexViewComponent 53

3.12 ACEView's ClassesByTagsView 53

3.13 ACEView's ACESubjectIndexViewComponent 54
3.14 ACEView's ACEDateIndexViewComponent 54
3.15 ACEView's ACEStakeholderIndexViewComponent 54
3.16 ACEView's ValidationView 55
3.17 The modified Ontology Definition Metamodel 56
3.18 The XML Metamodel 57
3.19 The UML Metamodel 59

4.1 The GQM paradigm level division [BCR94] 62
4.2 The plugin's new Views 74
4.3 Built Domain Ontology part 1 75
4.4 Built Domain Ontology part 2 76
4.5 First Inconsistency shown at the Validation View 78
4.6 Conflicting concepts shown at the Protégé inferred hierarchy view . . 78
4.7 First Inconsistency 79
4.8 Second Inconsistency 79
4.9 Third Inconsistency 80
4.10 Fourth Inconsistency 80
4.11 Fifth Inconsistency (a) 81
4.12 Fifth Inconsistency (b) 81
4.13 Sixth Inconsistency 82
4.14 Eighth Inconsistency 83
4.15 Tenth Inconsistency 83
4.16 ABox Reasoning 84
4.17 Inferred Hierarchy (a) 86
4.18 Inferred Hierarchy (b) 87

4.19 Protégé Inferred Axioms View 88

4.20 Generated PIM for the interview with the Security Specialist 89

4.21 Generated PIM for the interview with the Health System Specialist . 90

4.22 Generated PIM for the interview with the Medical Specialist 91

List of Tables

4.1 The GQM Measurement Goals 64

4.2 GQM for Scope Definition Issues 65

4.3 GQM for Communication Issues 67

4.4 GQM for Requirements Validation Issues 69

4.5 GQM for Traceability Issues 71

4.6 Interview Data 72

4.7 ABox Inconsistencies 84

4.8 GQM for Scope Definition Issues - Metrics values 92

4.9 GQM for Communication Issues 94

4.10 GQM for Requirements Validation Issues 96

4.11 GQM for Traceability Issues 98

Chapter 1

Introduction

The success of a software system can be measured by the degree to which it meets its envisaged purpose. Software system's requirements engineering (RE) can be defined as the process of discovering such purpose, by identifying stakeholders and their needs, and documenting them in a way that is amenable to analysis, communication, and subsequent implementation [NE00]. RE activities aim at managing all the requirements-related knowledge. Commonly such knowledge is stored at an artifact called *Requirements Document*. The production of such document is commonly included among the most difficult activities in the software development process. An effective RE process raises benefits such as: preventing errors, improving quality of the final product and reducing risk [Bro87, DPVOD02].

All resources applied in building a solid RE process have shown to pay off, once studies conducted by renowned IT consultant groups like the Standish, the Gartner and the Forrester groups pointed that a large number of projects fail to achieve their goals and some of them are even canceled due to requirements issues [GBLC05, GPFLC04, PWCC95].

The following sections will present a set of the most common problems encountered during software development processes and a set of related searches that have also pointed such problems. We will also discuss briefly the approaches proposed by such works to deal with these problems. After that we will present our main goals while developing this research project. Finally we will present the strategy that we have addopted to acheive our goals.

1.1 Problems in RE Processes

The importance of addressing RE activities has increased in the last years. Nowadays, the failure of a software system is no longer considered only as a programmer's fault; instead, there is a growing trend to discover the outermost reasons for such failures at software building processes.

Companies are investing in researches in the area of requirements management. Researchers are especially interested in identifying the most significant drawbacks of the existing requirement models and processes. Some of these researches, like [ARE96, BGMT06, GL93, CK92, NZP04] have already given some clue on what is wrong with existent RE processes. Following we will show a list of major open issues pointed by the authors, then we will discuss a little bit about each one.

These are the most common issues:

1. The need for a process to define the **requirements scope**;
2. The need for supporting for **communication among different teams** involved in the RE process;
3. A way to deal with the inherent **volatility** of requirements; and
4. The need for a traceability scheme to help isolating problematic requirements in further process' phases.

The correct definition of the system's limitations and scope is a very important issue in RE processes, since, most of the time, the client does not know exactly what he/she needs. Nowadays, such definition is up to the requirements engineer's, that has to do an interpretation of the interviews with the client. During this interpretation process the requirements engineer can lose himself/herself among useless descriptions given by a confuse stakeholder [Pre05, CADW01].

After collecting requirements knowledge from the client's stakeholders it is important that the requirements engineer can communicate it to other members in the requirements and development team. The appropriate notation(s) to represent such knowledge is then, crucial to ease communication between these teams. It should meet standard requirements for the RE document presented in Section 2 in order to better accomplish its goals.

Works like [SP90, MFKH90] established that requirements are volatile and that evolve along the time, so the requirements evolution and volatility is a great concern in software processes. Commonly requirements evolve due to the knowledge

brought up by the development activities or they are shifted because of unforeseen organization or environmental pressures [CK92]. Requirements also change over time because they are collected from several different sources (typically interviews with client's stakeholders), which may have different or even contradictory system's points of view [DHR89]. It is up to the RE process to manage such changes in requirements, minimizing their impacts at other artifacts.

The traceability issue is directly connected with the other 3 issues and any initiative to deal with such issues would have direct impact on the established traceability scheme used at the ongoing software development process. The RE traceability is concerned with relating requirements with other system's artifacts and it allows to follow the life cycle of a requirement artifact both forward and backward throughout a software development process [GF94, RJ01]. Such links between artifacts should allow the recording of meta-data about the RE process [GF97], like the stakeholder's name, who was the interviewer, etc. Such capability is very desirable to RE processes, since they are directly related to software maintenance and evolution [CHCC03]. A traceability scheme can help isolating requirements changing reflexes in later development process' phases, thus, reducing the costs and impact of requirements volatility.

Traceability schemes are deeply affected by the notation(s) chosen to represent requirements during a software development process, so, the choice of notation(s) should take the traceability scheme in account. Such concern gets greater when we are dealing with several integrated views with several notations to fully represent requirements. For example, if we choose to use a certain notation to represent the static part of the sytem, and an entirely different notation to represent the system dynamics. Then we should look for traceability schemes that suits both notations. We should be able to keep the requirements trace dependencies (i.e. the element that links two artifacts) updated throughout all integrated views[Egy03].

There are many research efforts in the RE area seeking answers to overcome these open issues. These efforts are concentrated in:

- defining processes to collect the client's knowledge in order to solve scope issues like [DMO$^+$06, BLC$^+$03, WEC93];
- defining models to represent requirements in order to address communication and traceability issues like [vL08, BdPL03, KS06]; and
- working on mechanisms and processes to be applied to requirements modeling in order to facilitate requirements evolution and maintenance, thus addressing volatility and also traceability issues like [Egy03, RJ01, CHCC03].

We have analysed these efforts and their proposed solutions in order to categorize them into a set of key-activities to be taken care off, during the production of the requirements document. Such set contains potential error-prone key-activities that, if executed without the necessary care, can lead to the occurrence of the requirements-related problems that are still considered requirements open issues. Following we will enumerate our classifications for the tasks to be executed during the production of the RE document:

- knowledge acquisition like what is proposed at [RW91, WEC93, WH96];
- knowledge representation like what is proposed at [MBJK90, JFH92, Gor04];
- knowledge conflict management and validation like what is proposed at [WJJ06, DLNS96, Len96]; and
- knowledge evolution like what is proposed at [CK92, CADW01, ELPA05];

This set of key activities show important points while defining a new RE process. These activities should be executed with special care, in order to avoid the occurrence of the mentioned RE problems. Following we describe how each of these key-activities is related to the occurrence of one or more of the cited problems in the RE process. Also, we present recent research efforts that propose ways to take car of each activity.

When the key-activity of knowledge acquisition is compromised, the RE process may suffer from scope issues, since the requirements may not be fully understood by the requirements engineer. To cope with this problem, several works propose methodologies to acquire knowledge directly from stakeholder's descriptions. The authors in [BdPL04, BLC$^+$03, DLM$^+$08, DMO$^+$06] suggest the employment of a restricted natural language (i.e. a natural controlled languages) to represent the requirements. The usage of this type of language allows requirements to be represented in a way closer to the natural human representation while keeping it free of ambiguities and imprecision. Also, the usage of such knowledge acquisition process helps speeding up and automatizing the collection of information from the client's stakeholders.

A misjudgment during the key-activity of knowledge representation may lead to a process that suffers from issues concerning the communication between development teams and stakeholders. It is a consensus in the RE area that the use of standardized notations to represent requirements can minimize the communication issues between software development teams and stakeholders. Besides, many works

[Whi04, KL08, vL08] argue that the employment and integration of different notations is necessary to represent requirements according to the different stakeholders' point of view, specially [vL08], where the author proposes the use of multiple views, linked through inter-model links constrained by rules for structural consistency. These works emphasize the importance of the use of several views and notations to fully represent requirements.

Researches conducted on the key-activity of conflict management and knowledge validation propose ways to assure that the requirements are not conflicting with each other and that they meet all clients demands. Activities of conflict management are very important to the RE process, specially because of its inherent volatility i.e. requirements are always evolving, and it is more likely that conflicts occur while including new requirements.

A failure during the validation of the new requirements may lead to the occurrence of problems related to volatility management issues at the RE process. Works in the area of requirements validation commonly use formal logic and inference mechanisms to check the consistency of the requirements and to assure that they are not in conflict with each other. Following this idea, [WJJ06, KS06, Len96] proposed the use of first order logic and ontologies to specify and validate knowledge requirements models.

Knowledge evolution activities are vital to the RE process since requirements naturally change over time[CK92]. Such activities are also related to the occurrence of volatility and traceability issues in the RE process. According to [CK92, MFKH90] this issue should be addressed by an iterative and incremental process. The amount of damages caused by a problematic requirement discovered at the very beginning of the software process is dramatically smaller than a missing functionality discovered after the delivery of the final product [MAJP02].

Both conflict management and knowledge evolution key-activities are deeply related to the traceability scheme. Such scheme minimizes the impact of the evolution of knowledge and it helps delimiting and identifying what are the affected requirements in conflicting situations. The biggest challenge while trying to implement a traceability scheme is to maintain the trace dependencies between the artifacts [Egy03]. Works in the traceability area are concerned in proposing (semi-) automatic processes to keep the trace links updated like the *scenario based scheme* in [Egy03] or the *event-based scheme* proposed in [CHCC03]. Researchers are also concerned in defining ways to trace the artifacts contributors like the *Contribution Structures* proposed in [GF97].

During this work we have developed a RE process that copes with all the key-

activities minimizing the occurrence of the RE open issues. The following Section explains our goals in details, describing the addopted strategy to specify each key-activity.

1.2 Main Goals

We have established one macro goal to be achieved during the development of this work. We intended to deal with all the open issues presented at the earlier section. We attacked each open issue by specifying the notation to be used at each key-activity responsible for generating such issue. In order to do so we assumed that there is no way to fully represent requirements by using a single view. Such principle has already been pointed by [SdPL06].

So, we needed a simple way to integrate the several views involved in our requirement specification solution. We have adopted a model-driven based approach. Its model transformation capabilities are fully aligned with our needs and may be used as means to seamlessly synchronize different views.

We have studied the MDD [Mel04] as a solution for the specification of the knowledge evolution key activity once it provides mechanisms to help keep RE models up to date. Such mechanisms are called model transformations and once the RE model changes, the same transformation can be applied to the new model to update the derived models.

We have studied its abstraction levels and focused on the *Computer Independent Model* - CIM to represent the requirements document. We have chosen such abstraction level because of its similarities to the requirements document. Both do not have any specific computational need. Both will be used at later phases of a typical software process after going through a process that aims to lower the abstraction level and include computational specific needs.

In order to achieve our macro goal we have divided it in 3 smaller subgoals: to define a set of views to represent requirements according to the needs of different profiles (the client, the RE engineer and the development team) in order to establish the knowledge acquisition and representation key-activities, i.e. to define the notation adopted to represent the CIM; to integrate such views in order to establish the knowledge evolution and conflict management key-activities, i.e. to establish a set of MDD transformations to build a mapping mechanism between views and develop a mechanism for requirements validation; and to establish a way to verify the effectiveness of our solution while dealing with the issues previously presented.

1.3 Proposed Strategy

To address all the mentioned key-activities involved in the elaboration of the requirements document as well as the open issues related to each activity, we propose an iterative and incremental *Model Driven RE Process* [Mel04]. Its phase division is aligned with former legacy RE process, defined in literature [Pre02, Som01], although we have proposed new activities, which encompasses all the key-activities. We believe that requirements should not take computational aspects into account, so, we use the MDD's CIM abstraction level to represent knowledge gathered during the requirements engineering activities. The CIM will compose our RE document.

Our work is focused at the static representation of the system (system dynamics is outr of scope). We propose the division of the CIM model in two abstraction layers: the client's view and the development team's view. This division was proposed based on the fact that these profile have very different sets of needs. The first needs an easy-to-learn notation in order to better understand the requirements provided by the client. The second needs a high level of detail and a set of tools to manipulate the requirements knowledge.

We adopted a *Natural Controlled Language* - NCL to represent the requirements gathered during interviews with clients. This adoption aids the knowledge acquisition and knowledge representation activities. It provides a language close to the native language of the client making possible to discuss the requirements specification using a notation fully understandable by both requirements engineer and client.

We adopted ontologies to represent the development team's viewpoint. Such adoption deals with knowledge representation, validation and evolution key-activities. We believe that this representation should be obtained from the system's descriptions in NCL. In order to keep both views synchronized we have defined a set of model transformations, used to automatically build the development view from the client's viewpoint.

Even though we focus on the CIM model, the proposed process ends with the building of a *Platform Independent Model* - PIM, to which we suggest the adoption of UML's Class diagrams as the main notation. We believe that such artifact can be useful at further development phases. The proposed PIM also aims to standardize the communication through all the software development process. Such communication can be eased by the addoption of model driven approaches

The remainder of this work is organized as follows: Section 2 presents the needed background to understand the other sections. Section 3 presents our designed process

an tool in details, Section 4 presents our case study. At Section 5 we listed a few related works and discussed the difference between such works and ours. Finally, Section 6 presents our comments about the obtained results and discuss our plans for future works.

Chapter 2

Background

2.1 Requirements Engineering

A software process defines how a software building is managed, supported, improved and organized [DKW99]. Every software development project uses a software process, be it implicit or explicit. In the worst scenario, the *ad-hoc* "methodology" is also considered a software process, an unordered and non standard process, but still a software process.

Software process are typically divided into *life-cycles*. The software life-cycle defines how a process should be conducted from start to end. The main goal of a software life-cycle is to build and deliver a software that meets, partial or completely, the client's needs [Sin96]. Typically a life-cycle is divided in phases and each phase is driven by a set of activities that produces a set of artifacts. Each activity can have an input and output set of artifacts.

One of the greatest challenges in a software process is to identify the client's real needs (software requirements). Some researches has shown that a large number of big projects fail because of invalid or misunderstood requirements [Boe84, GL93]. These researchers make clear that the main measure for a software project success is how much of the clients needs were fulfilled.

This tendency has also been described by [Som01, Pre05]. The authors argue that the activities developed during initial phases of a software process have great impact in the final product delivered. So, activities executed at the initial stages of a software process are crucial for the software development and its later success.

In that context, *Requirements Engineering* - RE is the process of discovering of the software purpose, identifying who are the affected stakeholders and what are

their needs, then documenting such knowledge in a usefull way, so it could be used in later phases in a software development process [NE00]. More precisely, RE is the process of discovering, analysis, management and documentation of requirements [ELPA05, AMEL04, TBF+03]. Such process is driven by the execution of activities [Som01, Pre05], that aim to build a document that holds the requirements knowledge (client's desired functionalities).

Due to the implicit subjectivity in the RE, researchers have not came up with the silver bullet to solve the RE activities problems. There is no recipe that one can follow to acquire success in the execution of the RE activities. The figure 2.1 shows the details of the interaction between the RE phases and their artifacts. This figure was taken from [Som01].

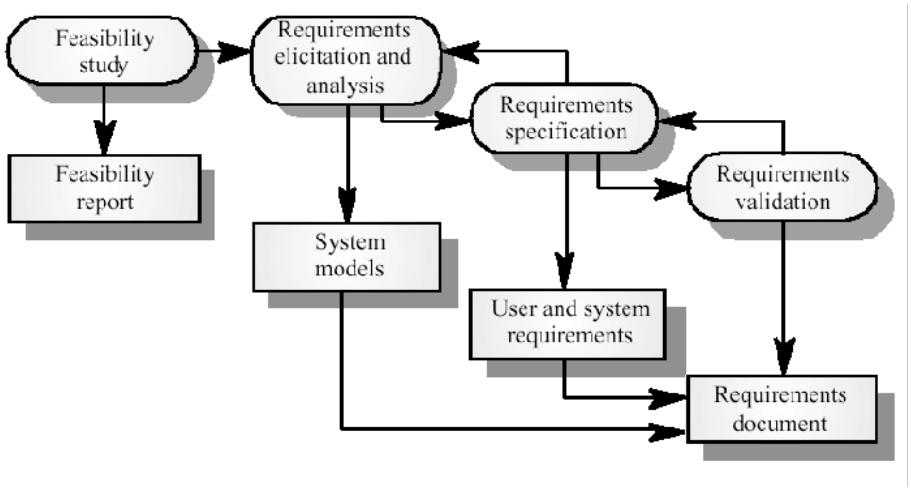

Figure 2.1: Requirements Engineering Process

2.1.1 Requirements Elicitation and Analysis

This phase has as its main goal, the production of a set of system models that should contain high-level descriptions of the client's needs. Such descriptions are build from interviews with the client himself or with one of his business analysts. The information needed to build the system models can also be enriched with the information contained in client's formularies and reports.

The interview are focused in discovering what activities of the client's business process should be implemented in software. This phase aims to collect information about the domain and the dynamics of the client's organization [Som01].

This phase has a viability report as its input artifact. This report contains information about the system boundaries and limitations. Such artifact is needed to

estimate more precisely deadlines and costs. It is the result of a viability study and should be taken in consideration by the requirements analyst before electing certain requirement to be implement, while building the system models.

2.1.2 Requirements Specification

This phase has as main goal to build requirements models. This models are detailed representations of the system models i.e. they have a lower abstraction level. The system models are its input artifact and the requirement model is its output artifact.

In order to build correctly the output artifact it is important that the requirements analyst, who specified the system models, be present while the process of detailing is done. The analyst should have collected all information needed about the client's business process, like involved entities, decisions to be made, flowcharts, etc. It is common to conduct new interviews during this phase in order to solve any remaining conflict.

The term specification has a lot of associated meanings in the software development context. All of them are related to ways of representing requirements acquired during the elicitation phase. So, one can conclude that there is a large number of notations to represent the requirement model artifacts [Pre05]. Among these ways we can cite: written documents, graphical models, mathematical models and interface prototypes.

The large number of notations to represent the requirement models turns the phase of requirements specification into a big concern, specially because there is no adequate notation to all software projects. Although, there is a set of desirable properties to the chosen notation [Gog93, GL93, ARE96, CK92]:

- It should be a computational model that could be used in (semi-)automatic process in order to build other artifacts that are useful during later activities;
- It should be able to hold usefull information for the later activities.;
- It should be readable by the client, so he is able to help the requirements analyst during the requirements validation activities, being able to certify that all his needs are fulfilled.
- It should be easy to maintain and evolute, having in mind that requirements specification is naturally an iterative and incremental activity and that requirements are constantly changing [CK92].

2.1.3 Requirements Validation

The requirements validation phase has as main goal to make sure that the specified requirements meets the clients needs. Naturally this phase should be conducted along with client. Its input is the requirements model, and the output is a requirements model approved by the client. The requirements model can then be added to the requirements document.

There is a set of techniques that can be applied in order to execute this phase, such as: *requirements revisions*, *prototyping*, *test-cases generation* and *automatic consistency analysis* [Som01, Pre05]. Among all of these techniques, the most used are requirements revisions. [Som01] explains that this technique is a manual process that involves a large number of the client's stakeholders. By consequence, such technique is very error-prone.

Once the requirements definitions should be inspected and agreed by the requirement analysis team as well as the client's business analysts, the notation becomes a very important choice. A clear and simple representation will help both teams go through the validation phase faster and less error-prone.

2.2 Model-Driven Engineering

The *model-driven engineering* (MDE) is the branch of engineering where the rigorous commitment to the model analysis is primordial. It tackles the problems of software development and evolution by promoting models to primary artifacts to be constructed and maintained [ALPT03]. The term was first used by [Ken02] and was inspired on the *Object Management Group*[1] - OMG's *Model-Driven Architecture*[2] - MDA, which is an initiative that developed a system architecture proposal based on modeling in several abstraction levels [GDDD04].

MDE has a broader scope than MDA, it has the philosophy of representing any piece of software in several models that are iteratively transformed in lower level models until they can finally be executed at a machine [ALPT03]. MDE is concerned with model construction and evolution. It tries to answer more philosophical questions like: How to build a model? What is the best way to persist models? How to establish mappings between generated artifacts? Such questions remained open in MDA's specifications.

[1] http://www.omg.org
[2] http://www.omg.org/mda

MDE is the result of a tendency in software development researches that intend to manage complexity through the usage of higher abstraction levels. In MDE, the software analyst tries to develop a solution based on the software functionalities and not in platform specifications. This tendency has been observed since the first compilers and programming languages e.g. the *Assembly* language was developed to protect the programming from writing machine code [Sch06].

The several abstraction levels are integrated by model transformation techniques. They are used to map certain model's elements at a higher abstraction level to lower levels. These model's elements are described by means of meta-models that are a set of meta-elements used to describe how the model's elements can be connected to each other.

The OMG's MDA specification establishes the existence of three main abstraction levels (or visions), that can be transformed from one to another linearly [Mel04] We have based this work on the existence of at least such three views at a software design process:

- The first abstraction level defines a *Computational Independent Model* - CIM, that is free of any reference to the computational concepts (system's data structures or low level algorithms). Such models can be seen as abstractions of the requirements models.
- The second abstraction level is the *Platform Independent Model* - PIM that is free of any specific platform detail (operating system, specific frameworks, persistence technology, etc.). It is focused on the relation between the entities, what data structure can be used to represent their relationship and what are the constraints that should affect such relationships.
- The third abstraction level is the *Platform Specific Model* - PSM, that is built after some technological decisions have been made (choose of operating system, programming language, data storage technology, etc.). It is one of the PSM's responsibility to merge the PIM with some platform and technological related aspects.

The integration of such abstraction levels can be seen in the following example: one client has given his description of the system, in that description he says that: *his domain is composed by students and books and that each student has a set of books.* Such description is found to be the project CIM's as it is.

During the PIM specification the designer designates that student and book are both going to be system's entities and there is going to be an unidirectional

relationship between them named *have* that parts from *student* and arrives at *book*. This knowledge can be represented by means of some standard modelling language like the *Unified Modelling Language*[3] - UML that is based on diagrams that represent different viewpoints of the system.

After that, the designer chooses to use a relational databases and designates that each entity is going to be represented by CORBA components. Such knowledge may be represented using UML too. Of course the diagrams would be enriched by means of one of the UML extension mechanisms. Suppose that a few time after that, finally people stop using CORBA. The client wants to migrate his/her system to a real enterprise framework, like Java's EJBs. Thanks to MDE transformations, the development team will be able to generate other platform specific models (EJB). The whole idea behind this example can be seen in figure 2.2 that was taken from [Fra03].

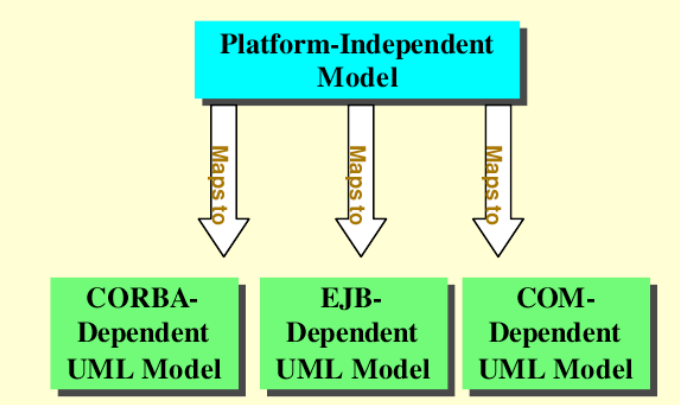

Figure 2.2: Several PSMs automatic obtained from a single PIM

The choice of what modeling language will be used to express each of the models in the system is very important and has great impact in which model transformation technique can be used to automatic build the lower abstraction level. The following subsection gives more details on how a modeling process is defined.

2.2.1 Models, Metamodels and Meta-Metamodels

Models have to be according to some modelling rules. These rules play an important role in model "syntax" checking. They are responsible for making sure that the model's semantics remains unviolated. Such rules are expressed in a meta abstraction level, called *Metamodel*. Commonly, metamodels are used to define modelling

[3]http://www.omg.org/uml

languages, like UML that is described in *Meta-Object Facility*[4] - MOF.

One of the most important concepts in modeling development is the idea of metamodel. It is in the metamodel that most of the model transformation approaches rely on. They manipulates the instantiations of a metamodel by means of the metaclasses. For instance: transformation rules applied to UML diagrams manipulate their elements by means of the UML metamodel where the concepts of classes, attributes, etc. are defined. But, metamodel are also models and also have to be described by means of some other metamodel. This third level of modelling is called meta-metamodeling.

It is not hard to see that this could go on forever in an infinite loop. To cope with such problem, the OMG's MDA proposes the definition of 4 model levels (from M0 to M3), where the last level (M3) is self-defined, which means that it uses itself as modelling language [MM+03, KBW03, MM+01]. The figure 2.3 illustrates how models, metamodels and meta-metamodels are organized.

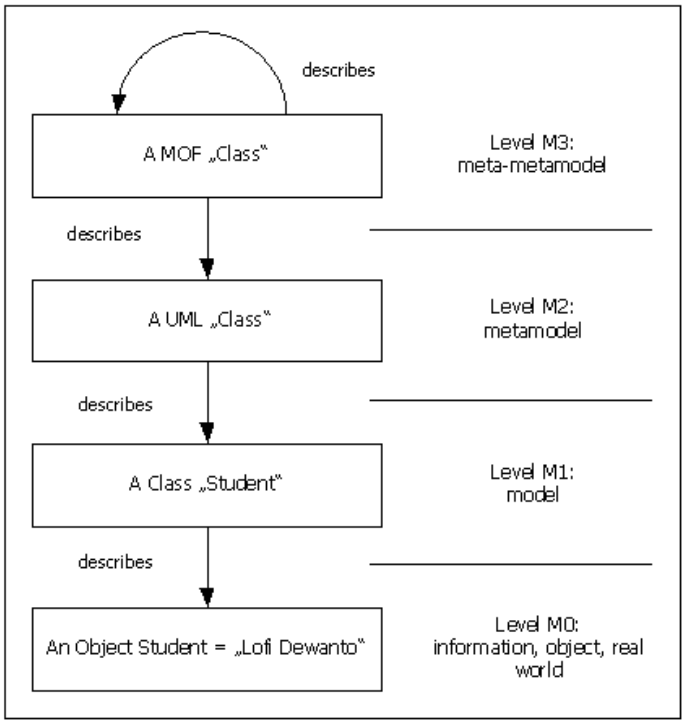

Figure 2.3: MDA's 4 layered approach

In figure 2.3 we have, at the **M0** level an object of the *Student* class, that has a name associated, it is an instance in memory. Such object was described in a domain model expressed in UML as a *Class*. The Class concept has also to be described. It was defined at the UML metamodel, that is expressed in MOF as a *MOFClass*. The

[4]http://www.omg.org/mof

MOF class concept has also to be described. It was defined at the MOF metamodel, that is also expressed in MOF.

2.2.2 Model Transformation

The whole model-driven-* idea is based on mappings between the abstraction levels, i.e. PIM, PSM, CIM. Such mapping mechanism is called model transformation technique. It is composed by the following elements:

1. Input Metamodel;
2. Decorator mechanism;
3. Transformation Procedures;
4. Output Metamodel;

The transformation procedure is based on the elements described in the input metamodel i.e. meta-elements. It maps such input meta-elements into another set of elements that are described in the output metamodel. The decorator mechanism provides the infra-structure to mark an input model element in a way that such element can be processed by an specific transformation procedure e.g. the UML profiles' stereotypes, which are applied to UML model elements, thus enhancing their semantics and aggregating some *meaning* to the elements.

MDA only specifies transformation from PIM to PSMs in a linear process and exclusively from PIM to PSM like what is explained at [Fra03, KBW03]. MDE on the other hand specifies that transformations should be used all over the model driven process. So, in the MDE context, one can define bidirectional mappings between any two abstraction levels. The figure 2.4 illustrates how the transformation process occurs in the MDA context, while figure 2.5 shows how the model transformation are defined in the MDE context

There are 3 main kinds of model transformation techniques:

- Programming based transformation;
- XML processing based transformation;
- Transformation languages based transformation;

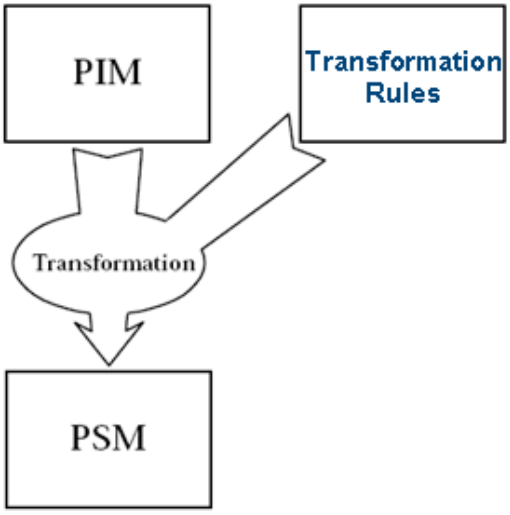

Figure 2.4: Mapping between PIM and PSM in MDA's context [Fra03]

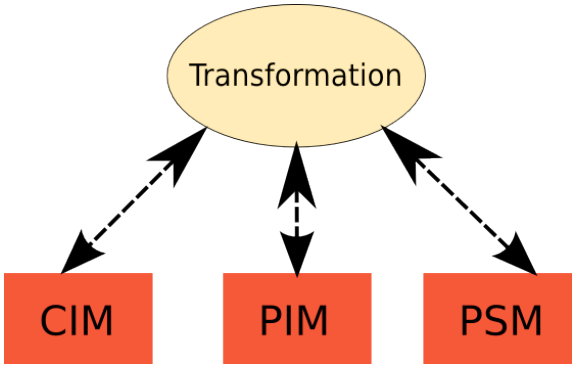

Figure 2.5: Mapping between CIM, PIM and PSM in MDE's context

One important element in the context of model transformation are the APIs for model manipulation. Such APIs are available at some popular programming languages like Ruby or Java. The later one has already very mature projects at the modelling area. Its main projects are available at the *Eclipse Modeling Project*[5]. Project that has given birth to the *Eclipse Modeling Framework*[6] - EMF, the most popular framework for model manipulation and code generation [Bud03]. Most of the transformation techniques use some specific model reading/writing API in order to manipulate the model elements.

In programming language transformation techniques, the API is in charge of reading metamodels and providing a manipulation interface to be used by the transformation procedures that are written by the the developer in some imperative

[5]http://www.eclipse.org/modeling
[6]http://www.eclipse.org/modeling/emf

programming language, e.g. Java.

The XML processing based transformation was largely used in the first MDA projects, since the OMG's recommendations for model persistence in file is the *XML Metadata Interchange* - XMI an XML with set of associated schemas. So, the best way to manipulate such format was the APIs for XML and processing like the *Document Object Model* - DOM [LHLHW+00] and XML stylesheet technologies, like *XML Stylesheet Transformation* - XSLT [C+99]. These techniques do not take in account the concept of metamodels since they deal directly with the XML elements. So, this is strictly a string manipulation technique.

The evolution of the XML processing based transformation included the use of transformation APIs to manipulate models. New template engines were also employed in order to reuse templates based on the metamodel elements. These modifications made possible the definition of semantics in transformations one they were developed having the metamodel in mind. An example o XML processing framework is the AndroMDA[7].

The metamodelling based technique uses special domain specific languages designed to manipulate models. The OMG has a propose for specification of such languages, it is called *Query View Transformation* - QVT which was designed to Query, View and Transform MOF based models [GGKH03]. Such specification has not became popular yet. Nowadays, most of the MDA projects use a language named *Atlas Transformation Language*[8] - ATL that is a declarative functional language for model manipulations [JK06] which implements partially the OMG's specification [JAB+06].

The ATL language is based on matching of elements of the input metamodel. Such elements are them used to instantiate output metamodel elements. Commonly, the ATL transformations are only declarative, but, the ATL designers have also provide some imperative extensions that can be used to write tricky transformations [JAB+06]. Each transformation procedure is called rule

The listing 2.1 shows an example of transformation rule that takes an Author model (that uses the author metamodel) and builds a person (that is described in the person metamodel). Such transformation only has one rule and that rule does nothing but a copy of the attributes *name* and *surname* from the input model to the output model.

[7] http://www.andromda.org/
[8] http://www.eclipse.org/m2m/atl

Listing 2.1: Author to Person ATL Transformation

```
1  module Author2Person; -- Module Template
2  create OUT : Person from IN : Author;
3  rule Author {
4          from
5                  a : Author!Author
6          to
7                  p : Person!Person (
8                          name <- a.name,
9                          surname <- a.surname
10                 )
11 }
```

2.3 Ontologies

Ontologies have been defined and used in a large number of areas, like in philosophy, as the study of the kinds of things that exist. In the context of system development they are commonly classified as an *Artificial Intelligence* - AI theory, a **content theory**, which describes objects, properties and the relationship among these concepts [CJB99]. This theory uses *First Order Logics* to build and represent itself.

There is a large number of languages that can be used to represent ontologies' classes and properties. In the latest few years, *Description Logics Systems* - DLS have been studied as formalizations for these languages. DLSs allow one to represent a certain domain of knowledge by means of concepts and roles, where concepts models classes of individuals (objects) and roles model the relationship between them (properties) [Len96]. These formalizations are the most used in ontology tools.

Altough, the concept of ontology is not so direct. It has an intrinsic duality that can be seen at the definition given by [CJB99]. According to the authors, the term ontology may be defined as both:

1. A specialized representation vocabulary to a given domain or subject; and
2. A body of knowledge used to describe certain domain by using a representation vocabulary;

The first definition states that ontologies define the entities and their relationships. For instance: in the classic pizza domain ontology [ND01] we have the concepts of

Pizza and *PizzaTopping*, then we have the definition of a relationship (predicate) *have* between these two concepts. Then the authors defined a restriction R that states that every *Pizza* has one or more *PizzaTopping* i.e.

$$R = \{\forall x \in Pizza, \exists y \in PizzaTopping \parallel have(x, y)\} \tag{2.1}$$

The restriction 2.1 works as a constraint to the *Pizza* concept. It defines rules of how the concepts relate to each other. According to [Len96], the axiom 2.1 is classified into the *TBox* (assertion about the concepts) aspect of DLSs.

The second definition states that ontologies are able to store knowledge about individuals. In other words, the representation vocabulary expressed at an ontology provides terms with which we can define facts in some domain. For instance, at the same Pizza ontology [ND01] one can model a fact F that states that a *Mozzarela Pizza have Cheese* as its *PizzaTopping* i.e.

$$F = \{\exists x \in Pizza, \exists y \in PizzaTopping \parallel (name(x) = Mozzarela \ \wedge \ name(y) = Cheese) \rightarrow have(x, y)\} \tag{2.2}$$

The *Mozzarela* is an instance of the *Pizza* concept and the relationship between it and the *Cheese* (instance of) *PizzaTopping* is also an instantiation, it has an specific individual as image and another as domain of the have relationship. The axiom 2.2 can be classified into the *ABox* (assertion on individuals) aspect of the DLSs [Len96]. The ABox axioms are (partial) instantiations of the TBox axioms that [DLNS96] relate individuals to classes (partial instantiation) or individuals to each other.

Although the difference between ABox an TBox axioms is very subtle, as both can compose an ontology, understanding this duality of the ontology concept is crucial so one can understand their role in system development. Bottomline, ontologies are used to model both, rules and facts. The first ones define the structure of the knowledge. The second ones are built according to the first ones and can be seen as instantiations of the first ones [DLNS96].

The process for building ontologies have changed along the years. First they were commonly built using AI techniques. Furthermore, the AI's 1990's approaches for building ontologies have converged with other area's approaches, like database designing and object oriented programming [Gor04]. This convergence was a result of the needing for representing knowledge and, specially, the needing for interchanging

information among all these research areas. This demand has given birth to a new research area, called *Knowledge Representation* - KR, also known as *Knowledge Engineering* - KE which is the application of logic and ontologies for the task of building computational models for certain domain [Gor04]. Most of the KE researches are focused in developing and testing methodologies for building and sharing Knowledge Bases in several areas.

Nowadays, ontologies are still widely used in AI researches, but also at the KE area [GPFLC04, GDD05, MGG+07], which has been largely used in system development projects. For instance, they are used as techniques for integrating and building web services in the context of the Web Semantics Initiative [Knu04].

The use of ontologies in system's development delivers a set of new desired features to the products, like [GDDS06]:

- The existence of a controlled vocabulary for the system's terms. This vocabulary should be ambiguity free and the contained terms may be used for communication between several applications;
- A DLS provides a reasoning capability for their descriptions. This reasoning process is composed by a set of logical operations that are able to identify inconsistencies and infer new relationships between classes (taxonomic classification). These new capabilities provide a clear definition of the relationships between classes (bringing up some hidden or intrinsic *is-a* relationships); and
- The use of ontologies provides the bases for knowledge sharing and reuse. To acquire this knowledge interchanging and reuse purpose, the applications should be able to interpret the ontology in the same way. This is assured by the formalization provided by the DLS.

One feature of this set stands out: the reasoning capability. By means of this capability one is able to: first, check the integrity of the TBox axioms, then classify the terms. The checking is done by a piece of software named *reasoner*. The checking identifies if any of the concept's constraints are broken. If this is true, the conflicting concepts may be considered inconsistent and, by consequence, they are disconsidered. The reasoning process can go on, discarding the conflicting concepts. The second step of the reasoning process builds the knowledge system's hierarchy tree (taxonomic classification). The invalid concepts are classified as subclasses of the "Nothing" concept [BvHH+04]. The ABox axiom set is also taken in account by the reasoning process in DLSs, although most of the actual reasoners like the *Pellet*[9]

[9]http://clarkparsia.com/pellet

and the *Fact++* reasoner[10] implementations can only check consistency and raise an exception if any problem is found i.e. some individual instantiation breaks some contraint defined in the TBox axiom set.

The reasoning feature can assure that certain common knowledge shared among several applications is free of inconsistencies. This is also very important while reusing pieces of knowledge and merging them with another pieces of knowledge, at a process named knowledge base merging, as described in [SM01].

OWL ontologies also have associated query languages, like SWRL [HPSB+04] or SPARQL [PhS+06], which provides powerful searching mechanisms.

Ontologies can be represented in several first order logic, like the *F-Logic* [KLW95] ("F" stands for frames), for instance. F-Logic is a frame based language designed to capture in a logically clean way a number of knowledge representation scenarios. The F-Logic although suitable to represent ontologies is still a logical language, a little bit hard to process and to understand by non logical specialists (programmers and system analysts, for instance).

Following the whole Semantic Web idea, a different language was proposed to represent classes (called resources) in a very generic way. This language is based on predicates and uses XML as its representation language. It is called *Resource Description Framework* - RDF [LS+99] and was proposed by the W3C web consortium. The following subsection explains the RDF technology along with its limitations and the *Ontology Web Language* - OWL, an specification based on RDF with enhanced semantics.

2.3.1 RDF(S) and the Ontology Web Language - OWL

RDF describes classes in terms of named properties and their values in an XML file (ABox). Along with the RDF framework, the W3C specification described the *RDF-Schema* - RDFS (TBox) as a mean to represent vocabularies to be used in RDF descriptions. The Schema approach also allows an easy way to validate an XML-RDF file. The listing 2.2 presents an example of how the before mentioned *Pizza* concept would be defined in RDFS.

Listing 2.2: Pizza RDFS Class

```
1    <rdf:Description rdf:about="Pizza">
2        <rdf:type rdf:resource="http://pizza.org/Class"/>
```

[10] http://owl.man.ac.uk/factplusplus/

```
3           ... other properties
4       </rdf:Description>
```

The XML language, although a little bit verbose is easy to process and can use already established parsing technologies and APIs. The main goal of the RDF(S) representation is to distribute its descriptions throughout the internet, using an webservice infra-structure [McB04].

The RDF(S) format, was found to be a little bit too general-purpose. It could only express part of ontologies. In the paper [AvH04], the authors list a set of necessary structures that are not delivered in the RDF(S) format. These are the structures:

- Local scope of properties: the `rdfs:range` defines the range to all classes, thus, in RDFS we cannot declare range restrictions that only apply to some class. The authors in [AvH04] have provided an example for this: take the *eat* property for instance. In RDF(S) it is impossible to state the predicate P that states that *cows only eat vegetables*, while other *animals may eat meat* as well i.e. $P = \{\forall x \in Cow \cap Animal, \forall y \in Food \parallel eat(x, y) \rightarrow y \in Vegetable\}$
- Disjointness of Classes: it is impossible to state that a class is disjoint from another. For example: Man and $Woman$ are disjoint concepts. In RDF(S) only subtype relationships can be described i.e. $P = \{Man \cap Woman = \emptyset\}$
- Boolean Combination of Classes: one may want to build new concepts by applying some set theory operation to other two classes, like the union, intersection or complement e.g. the $Person$ concept may be defined as the union of the concept Man and $Woman$ i.e. $Person = Man \cup Woman$.
- Cardinality description: sometimes the ontology designer needs to specify the amount of distinct values a property can have. For example, we would state that a $Person$ *has* exactly one $Mother$ i.e. $P = \{\forall x \in Person, \exists! y \in Mother \parallel has(x, y)\}$.
- Special characteristics of properties: sometimes we need to specify certain characteristics of properties the transitiveness of a property for instance like the smallerThan property, for example (i.e. $a \leq b \leq c \rightarrow a \leq c$), or the inverse of some property like the $isMotherOf$ that has the $hasMother$ as its inverse (i.e. $P = \{\forall x \in Person, \exists y \in Person \parallel hasMother(x, y) \leftrightarrow isMotherOf(y, x)\}$).

The OWL format was designed to supply these needs. It is an extension of the RDF(S) with an enhanced semantics and expression power. It is considered a richer

language but its expressiveness does not reduce its reasoning efficiency, once the richer the language the more inefficient the reasoning process is [AvH04]. Extremely rich languages reach levels that the reasoning crosses the computability border.

The OWL language is divided into three constructions subsets [BvHH+04]:

- The *OWL-Full* that holds all OWL constructs. It is extremely expressive, but this expressiveness has a price, it does not support full reasoning in their descriptions.
- The *OWL-DL* (DL stands for description logic) and its set of constructs ensure that the language correspond to a DLS, thus, supporting efficient reasoning. This is the most common subset of OWL and is supported by most of the ontologies authoring tools.
- The last set of constructs is called OWL-Lite. It is a simplification of the OWL-DL (excluding disjointness properties, properties cardinality among other things). It has as main advantages its simplicity to users and developers. This simplicity is also a disadvantage because it reduces considerably the expressiveness as a side effect.

The OWL authoring support counts with a large number of tools, like the Protégé[11], Swoop[12]and JOE[13]. This project has used the Protégé editor because of its popularity and source code available. The Protégé tool in its later version (Protégé 4.0) has a modular and highly extensible architecture based on events and a view plugin system that makes possible to write ontology visualization plugins that listen and respond to some ontology management related events.

2.4 Natural Controlled Language - NCL

Commonly, the human knowledge is expressed in Natural Language, which does not require any extra learning effort. This representations are most of the time informal, vague, ambiguous and potentially inconsistent [FKK08a]. A Natural Controlled Language - NCL[14] is a subset of the natural languages that can be processed by

[11]http://protege.stanford.edu/

[12]http://www.mindswap.org/2004/SWOOP

[13]http://www.cse.sc.edu/research/cit/demos/java/joe/joeBeta-jar.html

[14]We have choose to use the Natural Controlled Language term because it is very common in the literature, although it has an implicit paradox i.e. if certain language is natural then, by definition, it is not controlled and vice-versa.

machines. NCLs have constrained terminology, syntax and/or semantics. Still, its constructs are expressive enough to allow its intuitive usage by non specialists. Its concept is quite similar to the concept of a sublanguage, which is a language used in certain domain, commonly it has a set of domain-specific terms associated [Alt00].

A NCLs aim to reduce or completely eliminate the existing ambiguity, vagueness, informality and inconsistencies in Natural Languages [Alt00].They are composed by the same elements as natural languages: a lexicon and a grammar. The crucial difference between the controlled language and the natural language lies in the expression power of the language. Pure Natural Languages are slightly more expressive than controlled languages.

Controlled languages are largely used in commercial or industrial applications such as the authoring of user manuals, where large amounts of documents are generated and updated on regular basis and the document's terminology is domain specific [Alt00, NMH03], like the specification of the aircraft domain-specific maintenance language in [WHH98, WH96] or the *Caterpillar Technical English* - CTE described in [KAMN98]. They are also used in texts that might be translated to several languages, since they impose simpler constructs and, most of the time, ambiguity-free sentences.

In the computational area, NCLs have been widely used in the context of the Web Semantics Initiative to express knowledge about distributed resources like the PENG-D [ST04] that can be translated into first order logic fragments. The whole idea of replacing first order logic languages and formal methods with NCLs descriptions has become very popular in the latest years. This initiative is explained at [FST99, FKK08a].

In Software Engineering NCLs have been used to specify several parts of software (in several process phases and activities). Requirements knowledge specification in Requirements Engineering activities, for instance have been proposed by [FdSBdPL03, BLC+03, dSdPLB04]. Even procedural descriptions of systems have been using NCLs for its specifications like proposed in [LPCD06] to be used in construction (software coding) phases in software engineering process. Following we'll give some NCLs statements examples.

- *Attempto Controlled English* - ACE:
  - *Every man is a human.*
  - *Tom is a human.*

At this example we stated that there is a concept *man* that is a concept of the concept *human* i.e.*Every man is a human.* Then we described the individual *Tom* as an instance of the *human* concept i.e. *Tom is a human.* Following we will state the same sentences in *Common Logic Controlled English* - CLCE and *Sydney OWL Syntax* - SOS:

- CLCE
  - For every X, if X is a man, then X is a human.
  - There is a man X, such that X is Tom.
- SOS
  - If X is a man then X is a person.
  - If X is Tom then X is a man.

One can see that the same descriptions are done quite differently. In CLCE there is no distinction between individuals or classes. Such language is close to common predicate logic. The SOS language is part of an initiative that proposes NCLs to replace OWL, like the ACE language. The SOS altough close to a common unrestricted language uses a lot of variables, making the sentences harder to read and understand. On the other hand, the ACE sentences look a lot like non constrained language's descriptions. It can also use variables to express more complex sentences.

There is a large number of NCLS. We have chosen the ACE language to develop our work for its simplicity, intuitive learning capability and its tool set and editors. The next subsection will cover the ACE language.

2.4.1 *Attempto Controlled English* - ACE

ACE is a NCL with English syntax developed by the *Attempto Project*[15]. According to its authors [FHK+05, FSS99, FKK08a] ACE is so simple that can be used by people who do not know its restrictions. The authors relate that learning the construction and interpretation rules takes about two days.

ACE is, more precisely, a well-defined, tractable subset of full English that can be automatically and unambiguously translated into first order logic [FKK08a]. One could describe ACE as a first order logic language with an associated English syntax.

[15]http://attempto.ifi.uzh.ch

In fact, ACE's parser engine produces a variant of first order logic from the ACE sentences. Such variant is called *Discourse Representation Structures* - DRSs and can be translated easily into any formal language equivalent to (a subset of) first order logic (including description logic systems). It can also be used as an interchange format, making possible to share knowledge among several first order logic languages.

The following subsection covers in more details the ACE construction rules. It was based on [FKK08a]. Also most of the following examples were taken from [FKK08a], we have suppressed the citations for better reading. More details on ACE can be found at [FKK08b].

2.4.1.1 ACE construction rules

The vocabulary of ACE is composed of predefined function words (e.g. determiners, conjunctions, prepositions), some predefined phrases (*there is/are*, *it is true that...*) and content words (nouns, verbs, adjectives and adverbs).

The set of the ACE sentences is called the ACE Text. Each sentence can be either simple or composite. The ACE sentences are build using a simple set of *Construction Rules*[16]. In a similar way, the sentences interpretation is done by a set of *Interpretation Rules*[17].

The simple sentences are used to describe a fact, an event or a state. They have the following structure:

$$Subject\ +\ Verb\ +\ \{Complements\}\ +\ \{Adjuncts\}$$

The verbs complements are optional since they are only applied to transitive (*insert something*) and di-transitive (*give something to somebody*) verbs. The adjuncts (adverbs, prepositional phrases) are also optional. Following we will show examples of simple sentences:

- *A customer inserts 2 cards.*
- *The temperature is -2 C.*
- *At least 3 cards and exactly 2 codes are valid.*

[16]http://attempto.ifi.uzh.ch/site/docs/ace constructionrules.html
[17]http://attempto.ifi.uzh.ch/site/docs/ace interpretationrules.html

The nouns can have their descriptions enriched by the addition of adjectives. For instance, the sentence: "*A customer inserts 2 cards.*" can be enriched by the characterization of the subject *customer* and the object *cards* e.g. "*A new customer inserts 2 valid cards.*".

Composite sentences are build from simples sentences. To compose sentence one can use: coordination, subordination, quantification, and negation. Following we will give several examples of each type of composite sentences:

- Coordination by *and* (phrases **must** have the same syntactic type):
  - *A customer inserts a card and the machine checks the code.*
- Coordination by *or* (phrases **must** have the same syntactic type):
  - *A customer inserts a card or the machine checks the code.*
- *If-then* Subordinated sentences (conditional or hypothetical situations):
  - *If a card is valid then a customer inserts it.*
- Modality Subordinated sentences (express possibility and necessity):
  - *A trusted customer can insert a card.* (possibility)
  - *A trusted customer must insert a card.* (necessity)
- Sentence Subordination sentences:
  - *It is true that a customer inserts a card. (= A customer inserts a card.)*
- Quantification:
  - *Every customer inserts a card.* (universal quantification)
  - *There is a card that every customer inserts.* (existential quantifier)
- Negation:
  - *A customer does not insert a card.*
  - *No customer inserts more than 2 cards.* (negation of quantifier)
  - *It is false that a customer inserts a card.* (sentence negation)

The language also has query campability that allows to interrogate an ACE Text. Currently ACE only supports *yes/no-queries* and *wh-queries* (who, which, what and how).

The Attempto project provides a set of tools and APIs to manipulate and modify ACE Texts, including a semantic wiki (ACEWiki), an OWL Verbalizer, a Reasoner (RACE), a Rule Framework (ACERules), a Visualization Plugin for the Protégé Tool (ACEView) and a Parser Engine (APE). We will focus on the later two as they were largely used and modified during this project.

The ACE Parser Engine implements the ACE interpretation rules in Prolog. It uses a built-in lexicon of function and basic words of over a 100000 entries. This lexicon can be extended by users, who may want to upload domain specific lexicons. It takes user descriptions (ACE Text) and their domain specific lexicon (if there is any) as input and produces a large number of outputs, like the already mentioned DRSs. The figure 2.6 shows the APE web interface[18] with a sentence converted into a DRS representation. The APE is distributed with a LGPL license. It is also available through an WebService.

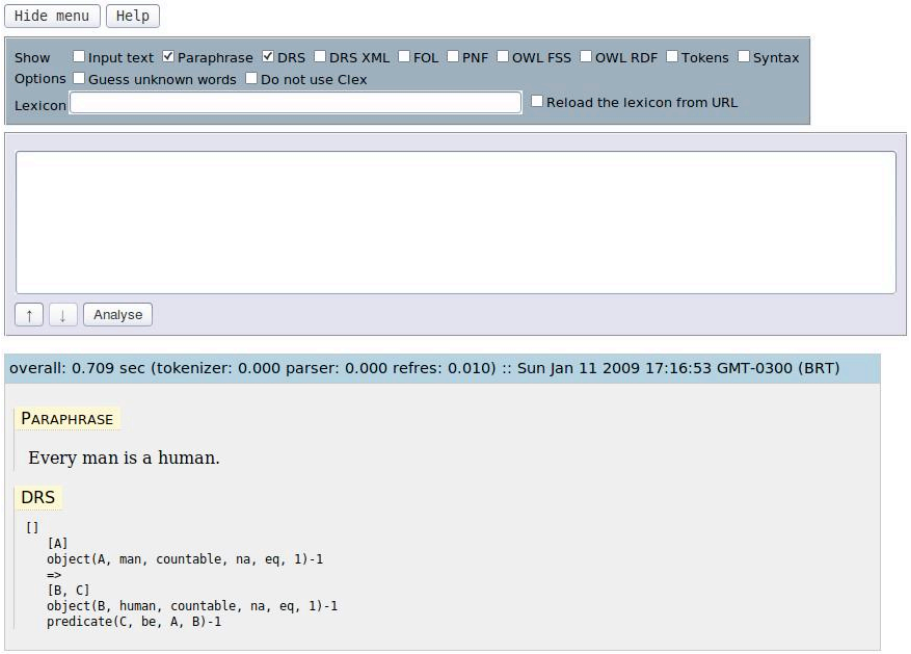

Figure 2.6: ACE Parser Engine WebClient

At our project, the most interesting feature implemented by the APE was the conversion from an ACE Text to an OWL file, like described in [KF06]. The listing 2.3 shows the OWL generate by the sentence: "*Every man is a human.*".

Another important part of the Attempto project was the ACE Text authoring tool: the ACEView Protégé plugin [Kal08]. It implements several views to better editing ACE Texts. These views are used for traceability purpose. With them one

[18] http://attempto.ifi.uzh.ch/ape/

can isolate each noun and relate it to each sentence where it was defined or cited. Figure 2.7 has a screenshot of the ACEView Plugin. The figure shows 3 of the several implemented views. The (a) view shows the division of the ACE sentences by content word, the (b) view shows the list of all snippets; and The (c) view is the ACE Snippet (sentence) editor, where one can edit each sentence.

Listing 2.3: Generated OWL file

```
1  <rdf:RDF
2      xml:base="http://attempto.ifi.uzh.ch/ontologies/owlswrl/test"
3      xmlns="http://attempto.ifi.uzh.ch/ontologies/owlswrl/test#"
4      xmlns:owl="http://www.w3.org/2002/07/owl#"
5      xmlns:owl11="http://www.w3.org/2006/12/owl11#"
6      xmlns:swrl="http://www.w3.org/2003/11/swrl#"
7      xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
8      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
9      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
10   <owl:Ontology rdf:about=""/>
11   <owl:Class rdf:about="#man">
12     <rdfs:subClassOf>
13       <owl:Class rdf:about="#human"/>
14     </rdfs:subClassOf>
15   </owl:Class>
16 </rdf:RDF>
```

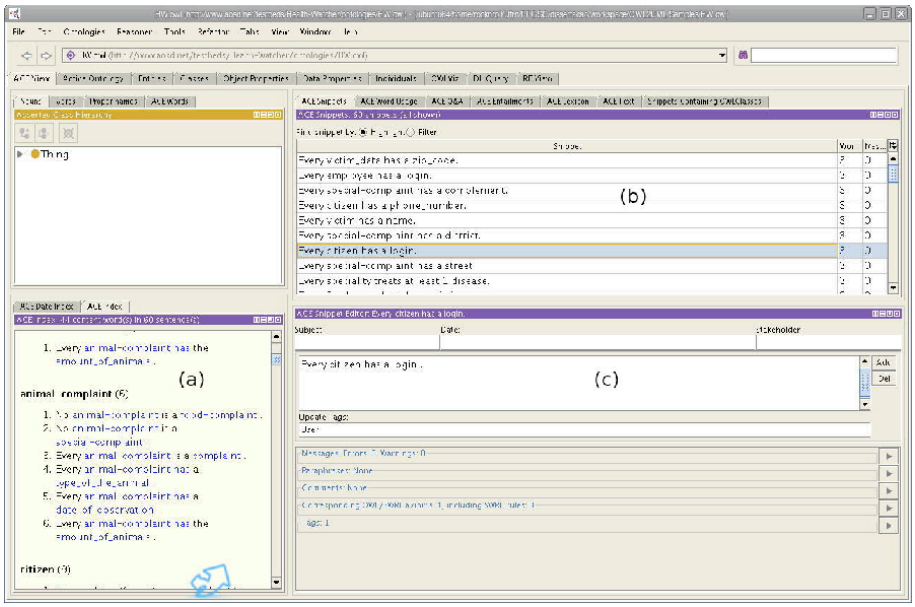

Figure 2.7: ACEView Protégé Plugin

Chapter 3

Proposed Approach

In order to address all mentioned key-activities (in Section 1.1) involved in the elaboration of the requirements document as well as the open issues related to each activity, we propose an iterative and incremental MDE based process to RE. The proposed process encompasses all the key-activities and focuses on the MDE's Computer Independent Model, since this model has the proper abstraction level to express the knowledge captured in the requirements document. The proposed process defines a CIM model split in two viewpoints: (i) the client's static view; and (ii) the development team's (static) view.

The following subsection explains our proposed models and shows how these models and views are used to address the other mentioned RE open issues (Section 1.1). Then we will explain the process step-by-step and finally we will present the project of a tool developed to help the process execution.

3.1 Proposed Models and Views

The adoption of our 2 layered representation for RE documents instantiates the knowledge representation key-activity (Section 1.1). We argue that, due to its complexity, the knowledge encompassed in RE cannot be fully expressed with a singled view model. To deal with this expressiveness problem we propose a CIM organized in two main abstraction levels, the **client's view** and the **development team's view**. During the build of the proposed CIM we also considered that each of these views can be naturally organized into **behavioral or dynamic requirements** and **static requirements**. However, representing dynamic requirements are out of the scope of this work.

In our approach, we have also proposed a mapping strategy between the client's and the development team's views. Such mapping facilitates the communication between the profiles involved at the software development process, thus addressing communication issues, providing a trustable communication channel between teams.

The proposed mapping strategy is also able to trace dependencies between views, making possible to follow the requirements throughout the development cycle, maintaining metadata about the requirements elicitation process (stakeholder's name, subject and date of interview, etc.). Such capability addresses the traceability issues. For instance, it makes possible to know, at a design phase, which stkeholder has given information about certain entity.

Although we focus on the CIM model, the proposed process ends with the building of a Platform Independent Model. Again, we are focused only at the static representation of the system. The PIM is composed by models that refine the development team's view models towards a detailed representation of the gathered requirements. We propose the usage of UML Class diagrams to represent the PIM.

Figure 3.1 shows the proposed subdivision in 2 views of the Computer Independent Model. Besides the CIM, Figure 3.1 shows our two PIM representation of the system. The following subsections will explain each viewpoint in more details.

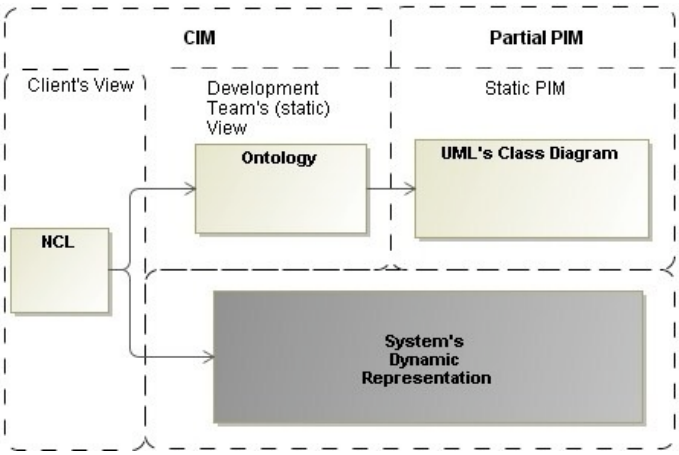

Figure 3.1: 4 Layered CIM and partial PIM

3.1.1 Client's View

This view aims to build a representation of the knowledge gathered during the RE process at an abstraction level that allows the client to understand such knowledge, thus facilitating his/her active participation in the requirements validation process. The main goal of this view is to enable the communication between the requirements

engineer and the client's stakeholders. To achieve this goal, requirements are represented in this view through a natural controlled language. This approach is similar to those proposed in [BdPL04, BLC+03].

The Client view is used to store the result of the execution of the knowledge acquisition and representation key-activities in the proposed RE process. It intends to represent the requirements of the system at the client's stakeholders point of view. Such knowledge is obtained by interviews. In order to build this view, one can also use a set of documents used by the client's employees during the execution of the client's business process, such as reports. Although, most of the time, these documents have to be aligned with the NCL restrictions, so they will have to be reviewed to use only NCL constructions.

Representing requirements with natural languages unburden the stakeholders of learning specific RE modeling notations, which commonly are not part of their expertise. Therefore, the adoption of natural languages facilitates the communication between stakeholders and requirements engineers minimizing potential misunderstandings and misinterpretations. In our process, we used the ACE [FSS99] NCL. The choice for an NCL based notation to represent the elicited requirements has a positive impact on the communication issues, since it provides a common communication mechanism between client's stakeholders and requirements engineers. More specifically, our choice for the ACE controlled language was based on its expression power, simplicity to use and learn and on the fact that it is also powerful enough to represent the fundamental concepts used to specify requirements, such as entities, relationships and restrictions.

Unfortunately, ACE does not have a defined traceability scheme, so in order to deal with traceability issues we have added the *tagging capability* (to trace dependencies) to the Attempto NCL. The tags are metadata used to mark sentences. It allows one to keep track of when a statement was given, who made that statement and what is it about (its subject).

The tags are kept through all the RE process. So, one is able to know what concepts were added by each interview. For example, one is able to know that the domain class *client* is related to the concept *client* that first appeared at an interview with the business specialist at December 10, 1966.

The inclusion of the tagging mechanism at the ACE NCL was done by extending its editor and its ontology metamodel, allowing the user to tag any sentence with anything he/she wants and persisting such information. The tagging mechanism is pretty generic and any kind of metadata information can be associated to the sentences.

3.1.2 Development Team's View

Commonly, the development team needs differ from both the clients and the RE team needs. Motivated by the works in [KS06, BdPL03] we enumerated the most common of these needs.

1. The need for a greater level of details in the requirements specification;
2. The need for a strategy to manage the complexity of requirements knowledge;
3. The need for tools that facilitate the access, authoring, reasoning, and searching of the requirement's knowledge;

Our proposed Development Team's View fulfills all these needs. The first and second needs are addressed by modularizing the requirements gathered in the client's view using different models that are expressive enough to refine such requirements and, at the same time, are in an abstraction level close to NCL.

The benefits of a modular representation are well-known from many areas of computer science and we use it, not only to support distributed development and re-use (as it is commonly done in software engineering), but mainly to manage the complexity of RE knowledge. Furthermore, modular representations potentially increase the efficiency of reasoning about requirements.

The third need is achieved by using models based on languages that are supported by standards bodies such as OMG and W3C[1]. Standardization efforts have been producing specifications that are broadly implemented as tools provided by several companies, many of them as open-source solutions.

For the development team view, we propose the use of ontologies, which allow a high-level and precise representation of a given domain in terms of its concepts and their relations. Moreover, it is possible to reason about a given ontology using available reasoning mechanisms. More precisely, we propose the use of OWL to model ontologies in this view. The adoption of ontologies to represent knowledge in development team's view addresses the key-activities of knowledge representation, knowledge evolution and knowledge validation.

Within our proposed process one can build this view in a (semi-)automatic way, having the client's view as input. This mapping addresses communication issues, allowing the requirements engineering team to communicate with the development

[1]http://www.w3c.org

team in a much simpler way, where the descriptions are (semi-)automatically mapped to ontologies.

The evolution and change of the knowledge stored at an ontology can use well stablished merging strategies, such as [SM01, SGP98, NM00]. The merging makes possible to reuse part of or the entire previous version of the requirements definitions (domain ontology), thus addressing volatility and evolution issues.

The use of reasoning mechanisms is crucial to execute merging tasks. Ontologies allow one to reason about its contained knowledge. The reasoning is composed by two activities. The first one is the consistency checking, where the reasoning mechanism verifies if there is any individual breaking any class constraint (ABox reasoning). For example: If it has been established that *only old clients can withdraw more than R$ 100,00 at an ATM*, the following logical expression formalizes such restriction

$$C = \{\forall x \in Clients\ \exists y \in Integer \parallel (withdraw(x, y)\ \wedge\ y \geq 100)\ \rightarrow\ x \in OldClients\} \quad (3.1)$$

Suppose that after that someone states that a *new Client has withdraw more than R$100,00 at an ATM*, i.e.

$$C = \{\exists x \in Clients\ \exists y \in Integer \parallel (withdraw(x, y)\ \wedge\ y \geq 100)\ \wedge\ x \in NewClients\} \quad (3.2)$$

Once we can assume that the $NewClients$ and $OldClients$ concepts are disjoint, i.e. $NewClients\ \cap\ OldClients\ =\ \varnothing$, one can infer from axioms 3.2 and 3.1 that there is an individual that has violated the disjointness constraint of its containing class.

The reasoning mechanism also verifies if any structural class constraint is broken (TBox reasoning), for instance, if any disjoint class have descendants in common. We will take the Pizza ontology [ND01] as an example it is a well known ontology example and its implementation is distributed with the Protegé OWL editor. At the example, the concepts of $VegetablePizza$ and $MeatPizza$ are disjoint i.e. $VegetablePizza\ \cap\ MeatPizza\ =\ \varnothing$. The concept $VegetablePizza$ is defined as "*Every Pizza that has a VegetableTopping*" i.e.

$$VegetablePizza = \{\forall x \in Pizza\ \exists y \in VegetableTopping \parallel hasTopping(x, y)\} \quad (3.3)$$

In a similar way, the concept $MeatPizza$ is defined as "*Every Pizza that has a MeatTopping*" i.e.

$$MeatPizza = \{\forall x \in Pizza\ \exists y \in MeatTopping \parallel hasTopping(x, y)\} \quad (3.4)$$

Suppose that after these definitions, one wants to define a $HealthPizza$ that is "*Every Pizza that has both VegetableTopping and MeatTopping*" i.e

$$HealthPizza = \{\forall x \in Pizza\ \exists y \in MeatTopping\ \exists z \in VegetableTopping \parallel \quad (3.5)$$
$$hasTopping(x, y)\ \wedge\ hasTopping(x, z)\} \quad (3.6)$$

Using axioms 3.5, 3.4 and 3.3, one could easily infer that $HealthPizza$ is both a $VegetablePizza$ and a $MeatPizza$ i.e $HealthPizza \in (VegetablePizza \cap MeatPizza)$ what clearly breaks our constraint that $VegetablePizza$ and $MeatPizza$ are disjoint concepts. Other constraint violations can also be revealed by the TBox reasoning process, like break of domain-image integrity, cardinality violation, etc.

The previous example showed the importance of using a logical representation for requirements. The adoption of ontologies deals with volatility issues, providing tools to check if the requirements base is free of inconsistencies. Such tools can be used when new requirements are added or when old requirements are reviewed and consequently modified. The usage of these tools assures that modified or new requirements don't lead to an inconsistent knowledge base.

The second activity executed by the reasoner is the taxonomic classification, where the mechanism uses the logical description of the concepts and their relationships and infer new relations between concepts.

The choice for ontologies also makes possible to keep our trace dependencies (knowledge tags). The tag information is stored at the ontology by means of their annotation capability. The propagation of trace dependencies (i.e. the NCL tags) is done automatically during the MDA transformation. The tags are stored in ontologies annotations.

3.1.3 PIM

Finally, the last step of our RE process, aims to build a PIM, which is also composed of two views, one that represents the static system's view and another that represents the dynamic system's view. Again, the specification of the system dynamics is out of this work's scope.

Our specification proposal does not cover a full PIM generation. Still, we propose to generate a PIM sketch composed of useful artifacts, that should help one to start, more quickly a design phase. We do not claim this PIM to be complete. It contains an UML class diagram, with the mapped entities and their relationships derived from the domain ontology. This PIM is not considered complete due to the fact that ontologies are not fully capable of keeping behavioral knowledge thus, our PIM does not contain any behavioral information about the system.

The transformation process responsible for the PIM generation is also able to keep the trace dependencies, making sure that the information provided during the client's interviews is still present at the PIM classes. Such information is used for traceability purposes, helping to group the PIM classes, by subject, for instance. To represent the requirements at the PIM static system's view we propose a transformation that derives UML class diagram from ontologies, by using a metamodelling model transformation technique. In this case, we developed the transformation using ATL rules.

This model transformation technique, as we have already said, is based on the idea of metamodels, where we should specify metamodels for the input and output of the transformation. OMG has already specified the UML metamodel.We have used the implementation provided by the *Eclipse Modeling Framework* - EMF framework. Also, to facilitate the employment of these model transformation techniques on ontologies, OMG has also defined the *Ontology Definition Metamodel* - ODM [CRH+06], a metamodel to manipulate ontologies through metamodelling transformations.We have used an ODM implementation in EMF provided by the ATL transformation park[2].

In Section 3.3.1.3 we describe the model driven transformation from ontologies to UML class diagrams developed in this work.

3.2 Proposed process

The RE process that we have proposed follows the RE process definition stated by Pressman [Pre05] and Sommerville [Som01] and it is divided in the following phases, which are executed in an iterative way:

1. requirements elicitation and analysis;
2. requirements specification; and

[2]http://eclipse.org/m2m/atl/usecases/

3. requirements validation.

Our process intends to keep compatibility with legacy software processes that use common RE approaches. And, although we have adopted the same phase structure proposed in the literature, the activities executed during each phase were modified in order to produce MDE artifacts by means of model transformations. Each phase is conducted by specific actors. The following subsections present the roles responsible for executing the process activities and the proposed process in detail.

3.2.1 Detailed Process

Each activity of our process is carried out by an specific actor. We have included new actors to the set described at Pressman [Pre05, Som01], specially the Ontology Engineer. The set of actors of our process is composed by:

- Requirements Engineer: collects information from the client and registers all information that is useful for the system modeling.
- Ontology Engineer: responsible for analyzing, merging and validating ontologies.
- System Analyst: selects which information is relevant while specifying and building a computational system.

The process with its artifacts is illustrated in Figure 3.2.

The first phase of our process is the requirements elicitation and analysis. The activities placed into this phase are conducted by the requirements engineer and his/her main goal is to gather system information from the stakeholders. Such activities are: (i) gathering information from client's stakeholders (activity 1, Figure 3.2), (ii) reviewing such information (activity 2, Figure 3.2), and (iii) specifying the system's behaviors (activity 3, Figure 3.2 - out of this work's scope). The output artifact of these three activities is a description of the system requirements in NCL, represented in the proposed RE process as the Client's definition of the system.

The information used during our ER process, like most of the software development process, is collected through interviews with the client (activity 1a, Figure 3.2) or by analyzing a set of documents provided by the client (activity 1b, Figure 3.2). As we agued in Section 3.1.1 such documents should be reviewed in order to get aligned with the NCL restrictions.

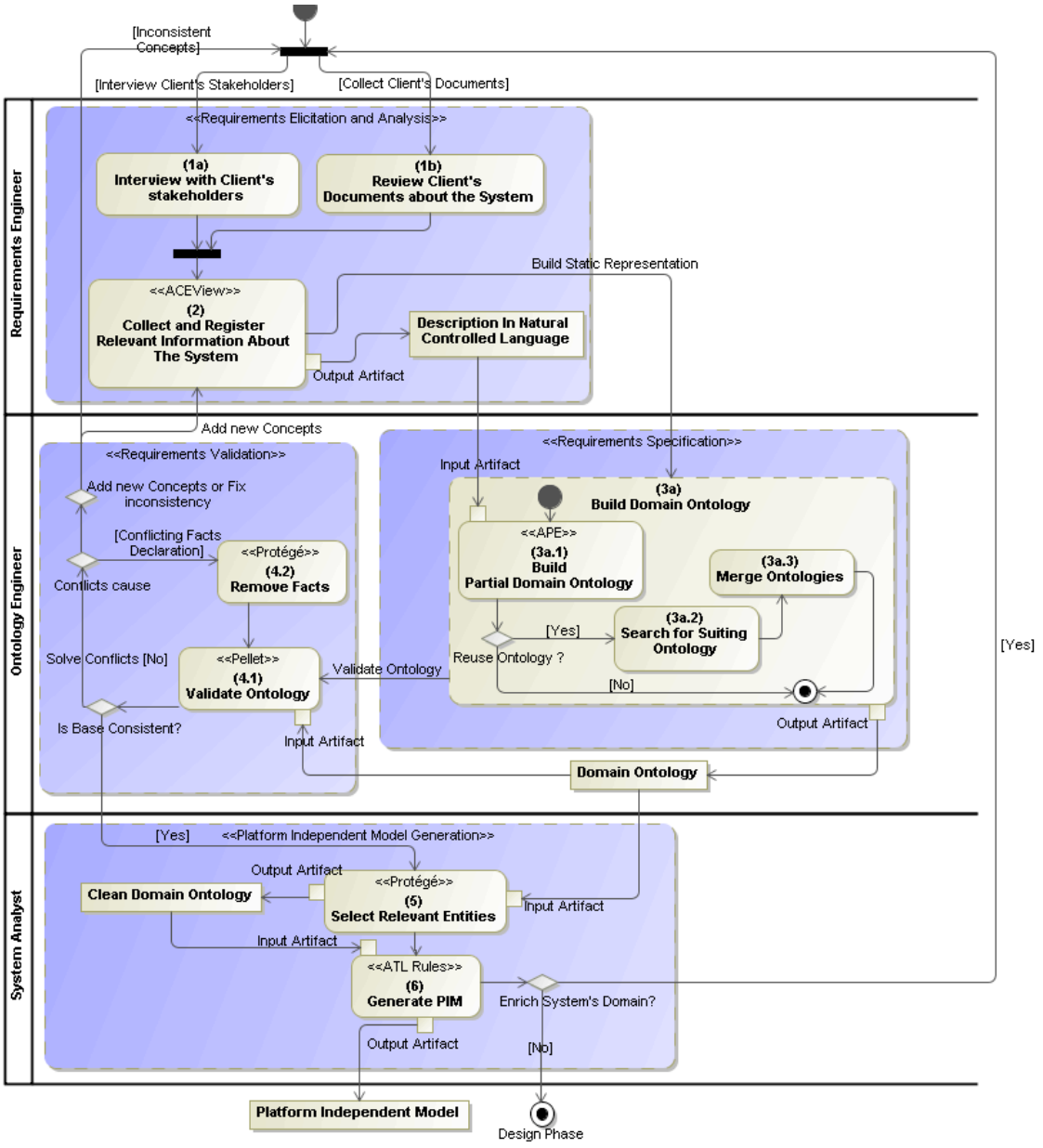

Figure 3.2: The proposed process (UML 2.0)

Often the interviews and documents provided by the client include unnecessary information, so the relevant information has to be initially filtered in order to build a consistent view of the system requirements. Activity 2 is responsible for accomplishing such goal. In this activity the requirements engineer builds a system's description in NCL. In order to provide traceability support, during this activity, the requirements engineer must include metadata information about the process of requirement acquisition, such as date of the interview, name of the interviewed stakeholder and subject of the interview. These metadata information is stored as tags at each sentence.

The second phase is the RE specification, where the models of the development team's view are built, i.e. the system's domain ontology, which represents the domain knowledge related to the system. The domain ontology model can be automatically derived from the NCL descriptions by an ontology engineer (activity 3a.1), by means of an MDE transformation. At our tool we used an MDE transformation written by the Attempto project to build an OWL ontology from our ACE NCL descriptions.

First he/she has to build a partial ontology, which encompasses all the client's provided knowledge at the previous phase. To do so, the ontology engineer makes use of an automatic ontology derivation mechanism, such as the Ace-to-OWL MDE transformation provided by the Attempto project, which we have incorporated into our process' support tool. Such transformation uses NCL as input and builds an OWL ontology from it.

After building the partial ontology, the ontology engineer can reuse some previously defined knowledge base (activity 3a.2), through an ontology merging operation. This predefined ontology may have been built on previous interactions of the process, by means of automatic NCL to ontology MDE transformations. This ontology can also be a well-known domain ontology used to represent a specific aspect of the system, for example: a monetary ontology to represent monetary concepts. Such reuse is not mandatory (however it is encouraged) and the process can go on without any ontology merging. In this case, the partial ontology is promoted to domain ontology.

By the end of the second phase, the CIM view will be complete and ready to be validated. The third phase of the proposed process is the the requirements validation that aims to find inconsistencies or conflicts at the requirements knowledge base. To accomplish this goal, we propose a (semi-)automatic approach that uses inference mechanisms implemented by reasoners. During the validation phase, two types of inconsistency can be found: inconsistencies between facts (individuals) and structural rules (concepts and relations) i.e. **ABox inconsistencies**; and inconsis-

tencies between concepts and relations constraints (constraints violations) i.e. **TBox inconsistencies**.

If the first case occurs, the inconsistent knowledge base becomes useless (it becomes locked) until the conflict is solved. This happens because most of the reasoners are unable to reason in this kind of inconsistent knowledge base. So, by consequence, the process also becomes locked, i.e. it enters at a loop and only leaves the loop if the knowledge base becomes unlocked again, which means that the problem is fixed. The process becomes locked because inconsistent requirements were discovered and any generated artifact from inconsistent requirements has no guarantee of being consistent.

In order to solve this kind of inconsistency, the descriptions written in NCL must be reviewed. The requirements engineer should look for stated facts that caused the inconsistency and discard them (activity 4.2, Figure 3.2).

For example, the statement "*Tom is a man.*" reveals the Man concept. However, the *Tom* (single and specific) individual does not tell much about the system's domain and once this statement has been processed, the concept Man has been added to the knowledge base, so the Tom individual can be removed without causing any further damage to the knowledge base. The track of the conflicting sentences is helped by the traceability scheme implemented at the supporting tool. After the validation of the ontology by the reasoner, the requirements engineer can track down the conflicting individuals and discover which are the sentences they belong to. After selecting the conflicting individuals and reviewing their conflicting sentences, the ontology engineer can successfully remove them. Suppose, for instance, that one has gathered the following information from a client's stakeholder:

1. *Every man is a human.*
2. *Every human has a last_name.*
3. *Tom has no last_name.*
4. *Tom is a man.*

After that, the ontology engineer fed the reasoner with the ontology generated by such set of sentences. The reasoner must answer that such ontology is inconsistent and as a matter of fact that's what it does. It presents the error message displayed at Figure 3.3 to the user.

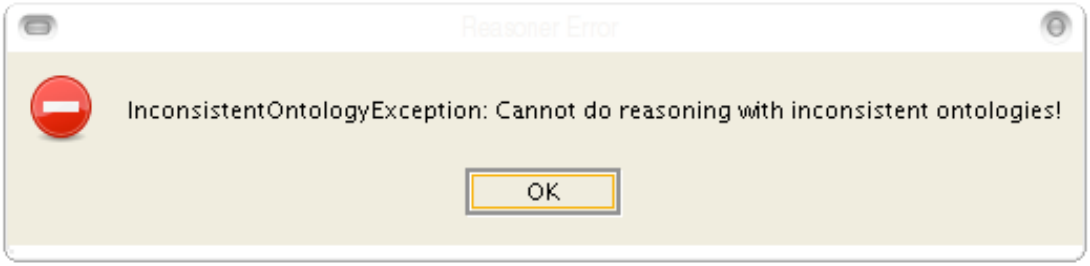

Figure 3.3: Message given by the Reasoner while trying to reason at an inconsistent ontology

After that, the ontology engineer can use the NCL editor to view where each individual is used, i.e. at which sentence the name of the individual is used. Figure 3.4 shows the NCL editor. One can see that the individual Tom, selected at the left panel was used at both sentences: "*Tom has no last_name.*" and "*Tom is a man.*". The origin of the conflict is in the fact that "*Every man is a human.*" and "*Every human has a last name.*", but "*Tom is a man.*" and "*Tom has no last_name.*". The tool then highlights the sentences in which the Individual Tom is cited, helping the ontology engineer to discover what are the conflicting sentences.

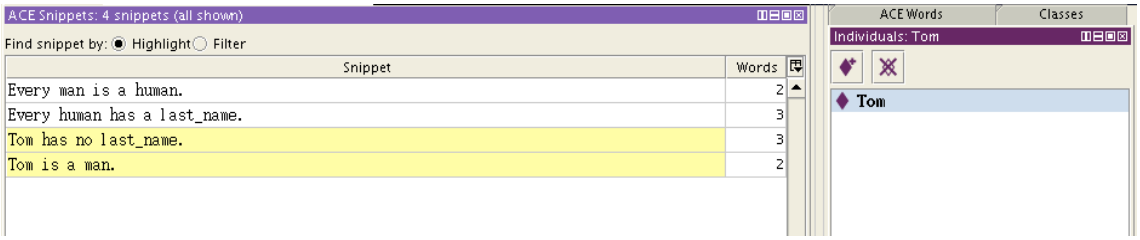

Figure 3.4: NCL editor showing conflicting sentences

Following our process, the ontology engineer would remove the Tom individual and re-run the reasoner, but at this time no error message is received. Also, no concept has been lost. And thanks to the traceability scheme one is able to track which conflicting individual is used at each sentence and vice-versa.

If the second case of inconsistency occurs, the requirements engineer should conduct a new interview with the client's stakeholder, whose descriptions resulted in inconsistencies. The identification of such stakeholders can be done by using the traceability scheme implemented at the supporting tool. It provides means to identify who is the stakeholder and when he/she has given the conflicting statement. The engineer can, additionally, collect a new set of documents. After solving the

conflict the process cycle should start all over, but at this time, the static knowledge already specified in the previous iterations can be merged with the new knowledge.

This kind of inconsistency allows one to continue adding knowledge to the requirements base (it does not become locked), but the conflicting concepts are invalidated. The reasoning process classifies them as subclasses of the *Nothing* concept, which is an special ontology class that allows its sub-concepts to be inconsistent.

Since the requirements base did not become locked, the RE engineer can add new knowledge to it. The invalid concepts should be treated by further MDE transformations (e.g the PIM generation transformation). At our tool, the sub-concepts of the nothing concept are just ignored during the PIM generation.

The use of the reasoner during the validation phase also has the power to discover new relationships of type is-a between the concepts, during the process of taxonomic classification. These relationships are built based on inference rules executed against the domain ontology's concepts (i.e. ontology classes) and their constraints that were defined by the NCL declarations and transformed into ontology's sufficient constraints. These constraints are typically built for sentences that start with "*Every*" or "*Everything*" and define new concepts from old ones or object properties. For example: suppose that at a medical domain scenario a stakeholder declared that "*Every person that has a disease is a patient.*", where the concept patient is defined in function of the has property, with disease as its image. i.e.

$$C = \{\forall x \in Person\ \exists y \in Disease \parallel has(x, y)\ \rightarrow\ x \in Patient\} \tag{3.7}$$

Afterwards, another stakeholder states that "*Every victim is a person and every victim has a disease*" i.e.

$$V_1 = \{\forall x \in Victim \parallel x \in Person\} \tag{3.8}$$

$$V_2 = \{\forall x \in Victim\ \exists y \in Disease \parallel has(x, y)\} \tag{3.9}$$

Thus, during the taxonomic process, the reasoner can infer, based on axioms 3.7, 3.8 and 3.9 that "*Every Victim is a Patient*" i.e.

$$P = \{\forall x \in Victim \parallel x \in Patient\} \tag{3.10}$$

The next activity of our RE process is the (partial) PIM generation, which is not commonly included within traditional RE processes. We consider this phase as a

transitional phase to the next development software phase. The goal of this phase is to automatically build a useful artifact for the design phase of software development life cycle. The PIM generation is carried on by a system analyst that executes model-driven transformation taking as input the RE knowledge bases generated in the developer team's view, producing the PIM. We believe that, at a MDD-based process, the PIM generation activity should also take the system's dynamic definition as input, although this functionality is out of the scope of this work.

3.3 Tool Support

In this section we will explain the architecture and functionalities of the tool that we have implemented. Also we will relate its parts and functionalities with the proposed process.

To support the proposed process we have used and extended a few tools. Each tool plays an important role at the process execution. We have modified our process activity diagram to show where and when each tool is used. The modified diagram can be seen at figure 3.5.

The first applied tool was ACEView and such tool is used as an NCL editor, so every interaction with the NCL descriptions occurs through the ACEView plugin. It is first used at activity 2, when the requirements engineer searches throughout the descriptions and documents provided by the client for relevant information about the system. We have extended the ACEView to add the tagging capability in order to implement our interview traceability scheme. These modifications have the purpose of fully aligning the tool with the proposed RE process. Our extensions enable grouping NCL statements by interviews, which represent the artifact generated as output of activity 2 in Figure 3.2. Also, the tagging feature provides traceability information, indicating which interviews brought each piece of knowledge up, thus addressing traceability issues.

The requirements engineer uses the editor to mark each NCL statement with a set of tags that identifies the interview to which the sentence belongs to. To encapsulate an interview we have proposed a structure called *Subject-Date-Stakeholder* - SDS, which is composed of information (tags) about the requirements elicitation process (date of interview, name or role of Stakeholder and subject of interview). For example: suppose that at April first, 2009 the requirements engineer interviewed a security analyst from a certain company in order to collect information about the user control subsystem. So, the SDS tags for this situation would be: "*User Control*", "*1/4/2009*" and "*Security Analyst*".

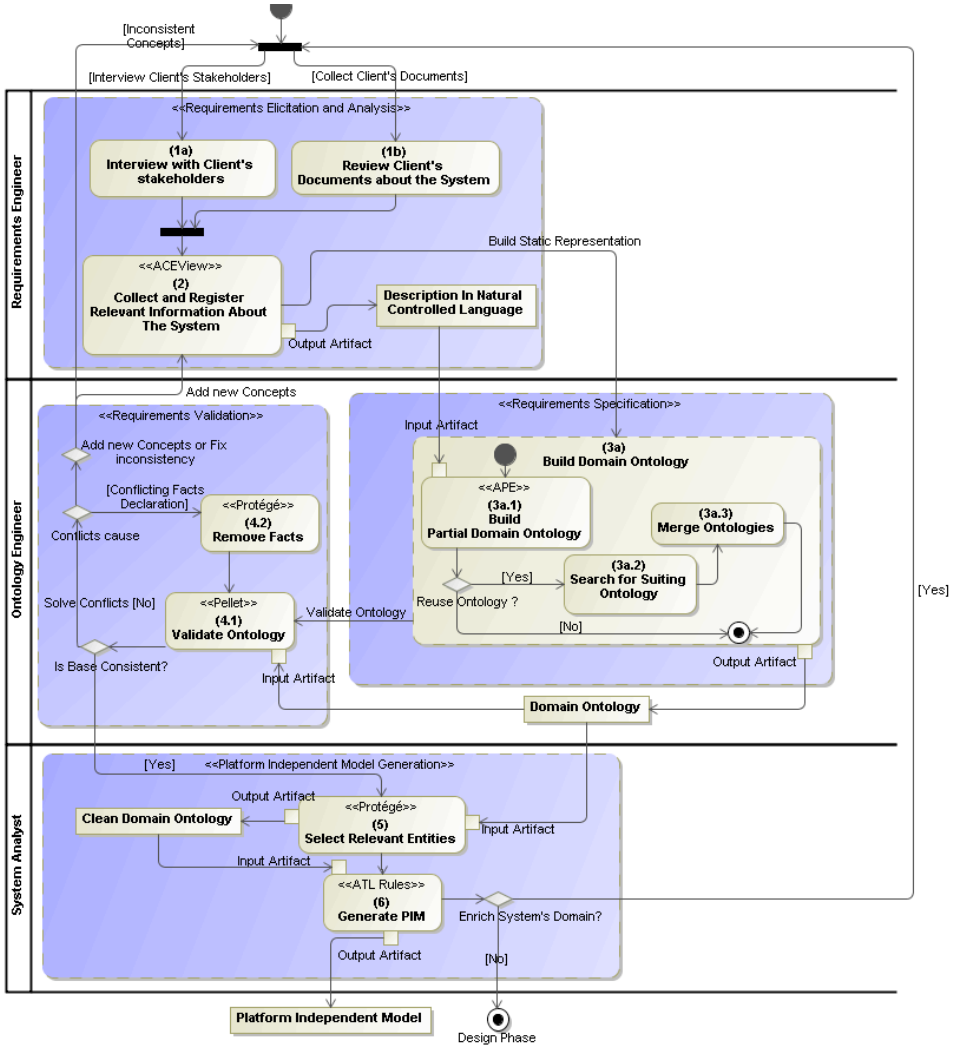

Figure 3.5: Proposed Process with specification of Used Tools

The SDS object must be filled for each sentence. The set of sentences marked with the same SDS composes a complete interview. The designed tagging capability is very extensible and generic. The RE engineer is able to include any kind of information into each sentence. Such design choice makes possible for one to extend our solution providing new capabilities, for example, making possible to add support for document storing, where such document can be provided by the client or a transcription or even the audio record of the interview process.

After the definition of the sentences by the requirements engineer, an ontology engineer builds the domain ontology from such descriptions. He uses the *ACE Parser Engine* - APE, developed and provided by the Attempto project to transform the textual description into ontology concepts and properties. This is the first MDE transformation addopted at our tool.

Assuming that we already have a domain ontology, we addopted the Pellet reasoner to validate it. The ontology engineer is able to use any reasoner he/she finds suitable. We also have made a few tests with the Fact++ reasoner and it showed the same results, so, as far as we have tested these two are interchangeable for the validation activity.

After the validation activity the System Analyst is able to select the relevant information from the domain ontology. We have chosen to use the Protégé editor for ontologies for its popularity, simplicity to use and source code availability. Also it is the base platform for the ACEView editor. It presents a simple to understand event-driven interface that helped us understing the ACEView design when we needed to modify it.

Finally, we have implemented a series of ATL transformations to build our PIM from the OWL ontology. Such transformations automatize the mapping between the CIM and PIM views. Since this mapping occurs at an automatic way, once our transformation rules are well defined, the same behaviour would be applied to every entity at the input model. The adoption of this automatic approach helps dealing with communication issues since it defines an uniform mapping for the entities. We have chosen the ATL model transformation language to implement these transformations because of its maturity level and active development community. This transformation based approach also helps dealing with traceability issues. We have defined our transformation in a way that no interview trace goes missing during the mapping process. The transformation rules guarantee that the PIM will also carry those traces.

Bottomline, the implemented MDE transformations map the NCL statements of an interview into a set of ontologies concepts (or entities) and their relationships.

Later, these ontologies are transformed into classes at the PIM class diagram. The transformation workflow was defined by using specific Ant tasks provided by the AM3[3] eclipse project. We have built an Ant build that reads the input models (OWL files and ODM XMI models), its metamodels and executes the transformation.

The Figure 3.6 shows a screenshot of our modified tool.

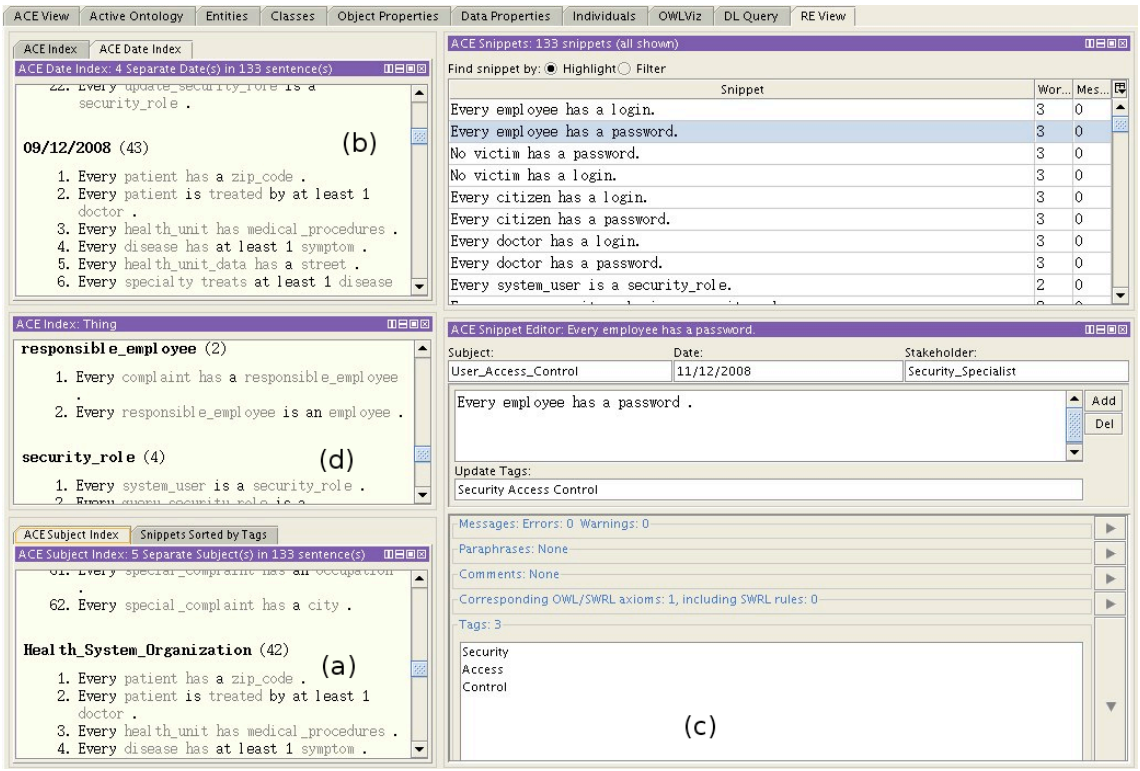

Figure 3.6: The New ACEView

3.3.1 Proposed tool architecture

Figure 3.7 shows the architecture of the implemented RE tool, which delivers the following functionalities to the users in order to provide support for the proposed process:

1. reading of descriptions written in NCL. The ability of reading NCL descriptions is important to provide support for activities 2 and also activity 3 at our process.

2. transform NCL descriptions into ontologies and validate them. This requirement makes possible to implement activity 3 of our process.

[3]http://www.eclipse.org/gmt/am3/

3. parsing ontologies files and building ontology models. This is the first step on the transformation from the CIM into the PIM. The ontology model represents the ontology by means of model description techniques. Such transformation is needed in order to execute the ATL model transformation rules. We had to make a little modification at the ontology metamodel, which describes the ontology itself. We have introduced the *Tag concept* at it. This functionality is required in order to implement the support for activity 6 of our process.

4. applying transformation rules to build UML models from ontology models. This is the second step needed to build the PIM from the CIM. Thus, it is needed to implement support for the activity 6 of our process.

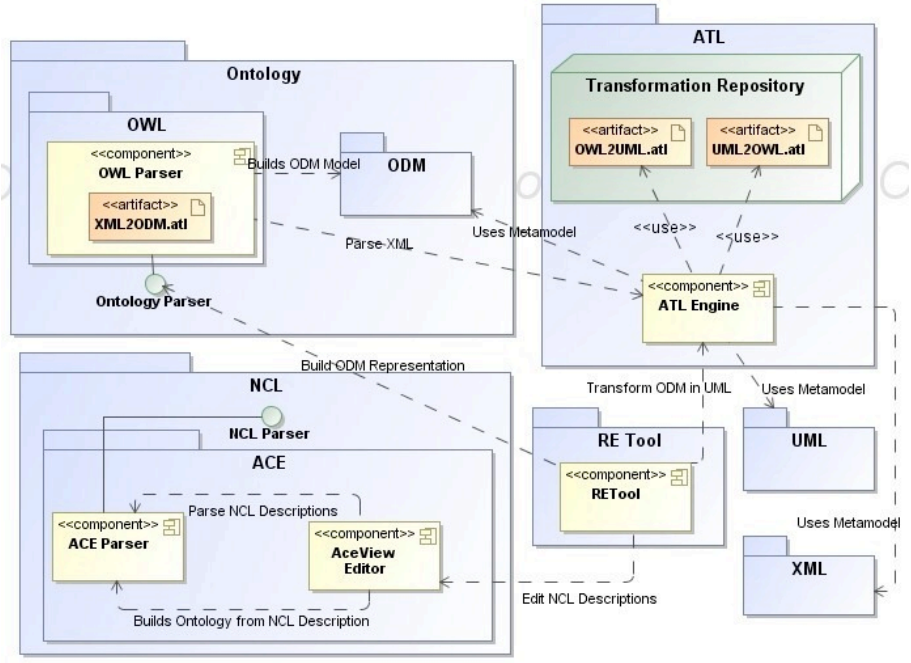

Figure 3.7: RE Tool's Architecture (UML 2.0)

The components in Figure 3.7 are organized in UML packages according to their responsibilities. Following we explain the responsibility of each package.

3.3.1.1 NCL package

This package provides features to edit and parse textual descriptions as well as build ontologies from such descriptions. It implements the required support to the activities 2 and 3 from our process. Its main components are the ACE Parser and the ACEView Editor plugin, which we have extended, by adding the tagging capability.

The plugin takes the clients NCL descriptions as input and uses the *Attempto Parser Engine* - APE for parsing them. The APE's main functionality is to execute

the first MDE transformation whose goal is to convert textual descriptions into OWL files. APE also provides a function that converts OWL ontologies into textual descriptions. Both functionalities are available via Web Services [FKK08b].

The NCL package functionalities are mainly used at the requirements elicitation and analysis phase (activity 2 in Figure 3.2.). It also aids the specification phase of our process, more precisely at activity 3a.1 in the building of the partial domain ontology. The language parser and the basic editor are provided by the Attempto Project[4].

In the proposed process, textual descriptions are produced by using the ACE language's editor of the AceView. The plugin defines a set of views that aid users to write their ACE descriptions. The editor is mainly composed by visualization and domain components. Figure 3.8 shows the main domain classes of the ACEView. The classes `ACETextManager`, `ACESnippet`, and `ACEText` had to be modified in order to implement the tagging of sentences feature. The class `SDS` was completely implemented in the context of this work the rest of the classes were implemented by the Attempto Project. Following we will explain each one of them.

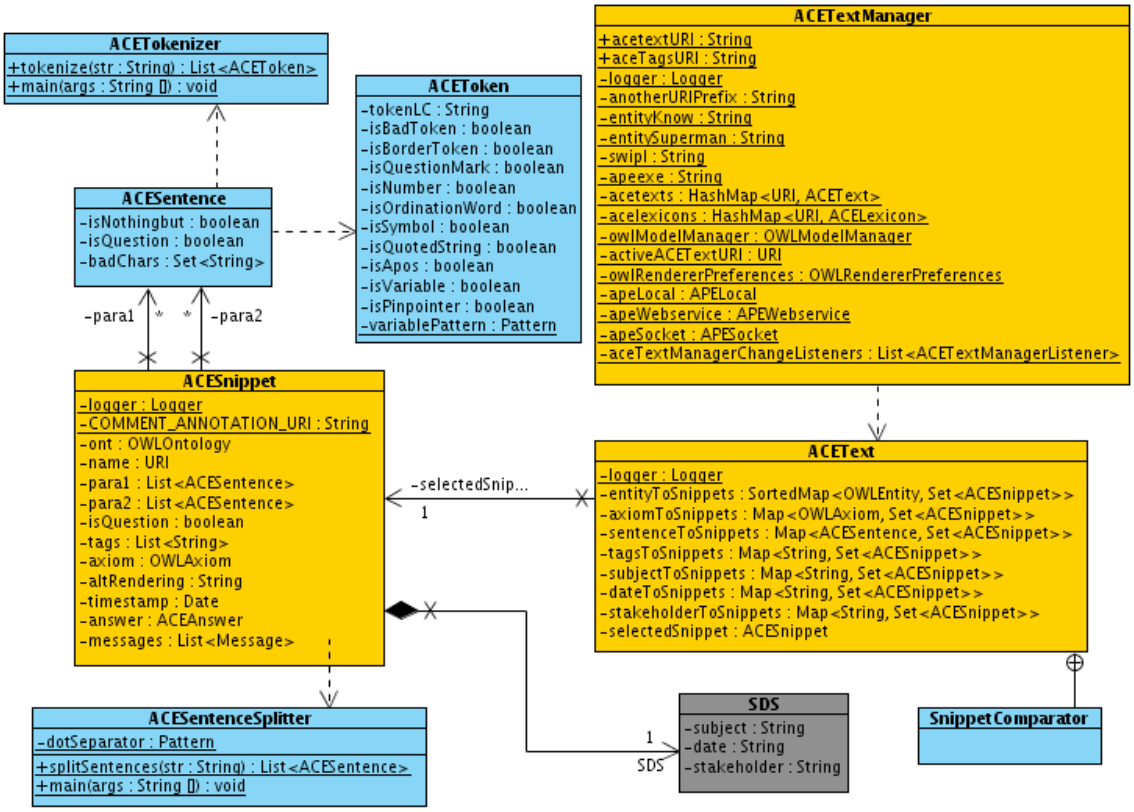

Figure 3.8: Main Classes of the ACEView Domain

The `ACESentence` class is used to record each sentence declared by the user

[4]http://attempto.ifi.uzh.ch

of the editor. This class is responsible for inspection operations on the Sentences objects, such as the `isQuestion()` operation that answers if the target sentence is a question.

The `ACESentence` stores a sentence as a set of `ACETokens`, which are the smallest part of a sentence and are built from a common *String*, by the `ACETokenizer` class. An `ACEToken` can be a verb, a preposition, punctuation or any of the elements described in Section 2.

The `ACESentences` are grouped into `ACESnippets`, which may contain a set of sentences given by the user of the plugin. The `ACESnippet` is important, since each statement given by every stakeholder may be composed by one or more sentences, e.g. the composite sentences presented in Section 2. The `ACESnippet` is composed by a set of sentences, and we added a set of tags and a SDS object, which represents the requirements elicitation process information (stakeholder's name, date and subject).

The `ACESnippet`'s set of sentences is built by the `ACESentenceSplitter` which receives a *String*, converts it into phrases, then, each phrase is used to build an `ACESentence` composed by a set of `ACETokens`.

The `ACESnippets` are grouped into an `ACEText`, which is responsible for storing all sentences, be them simple or composite ones. It maintains an instance of the current `ACESnippet` which is being edited (`selectedSnippet` property in Figure 3.8). To organize the snippets into the `ACEText`, it uses a comparator object, the `SnippetComparator`, which serves mainly to check if a new provided snippet has already been registered. Also, the `ACEText` maintains a set of Maps in order to group sentences by several criteria and optimize searches within the `ACESnippets`.

The `entityToSnippets` *SortedMap*, for instance, helps relating each one of the ontologies entities to the set of Snippets into what they have been cited. The same principle applies to the `sentenceToSnippets`, which maps a sentence to a set of snippets in what it has been declared. We added a new map to this class, the `tagsToSnippets`, which relates a tag to all Snippets marked with such tag.

The ACE parser engine has an additional feature that makes possible to paraphrase a sentence, which is another way to express the content of each sentence (composite or simple) by means of other sentences. In the diagram in Figure 3.8 this feature is represented by the two relationships between `ACESnippet` and `ACE-Sentence`, named `para1` and `para2`.

All operations related to `ACEText` edition are done by a service class named `ACETextManager`, which is responsible for accessing the parser webservice, storing

and finding within the `ACETexts`. Following we will enumerate the main operations of the `ACETextManager` and discuss its usage:

1. `createACEText()`: responsible for creating the `ACEText`. It receives an URI, provided by the user that identifies the underlying ontology;
2. `getActiveACEText()`: returns the `ACEText` which is currently being edited;
3. `add()`: adds the `ACESnippet` given as a parameter to the current active `ACEText`;
4. `remove()`: removes the `ACESnippet` given as a parameter from the current active `ACEText`;
5. `addSentences()`: adds a set of sentences (simple sentences), given as parameters, to the current active `ACEText`;
6. `removeSentences()`: removes a set of sentences (simple sentences), given as a parameter, from the current active `ACEText`;
7. `createOntology()`: creates the underlying ontology;
8. `createACETagAnnotation()`: the provided tags into the underlying ontology as an ontology *Axiom Annotation*;
9. `parse()`: parses the current active `ACEText` into an OWL file, by acessing the APE webservice;

We have also extended the editor with new views that help the traceability scheme of requirements and execution of MDE transformations. We have organized these views into a new Protégé ontology tab, called RE View Tab. Figure 3.9 shows a class diagram that lists the components of the new developed views. The class `ACESnippetEditor` was modified to add the tagging capability. The classes `ValidationView`, `REViewTab`, `ClassesByTagsView`, `ACESubjectIndexViewComponent`, `ACEStakeholderIndexViewComponent`, and `ACEDateIndexViewComponent` are entirely new components. The other class was provided by ACE and did not suffer any modification. Following we will discuss the purpose of each one of this views, along with screenshots from the tool.

1. `ACESnippetEditor`: this component is used to create, remove and edit `ACESentences`. It should also be used to enter the user defined tags and the SDS tags (subject, date and stakeholder's name). This view is composed of a list

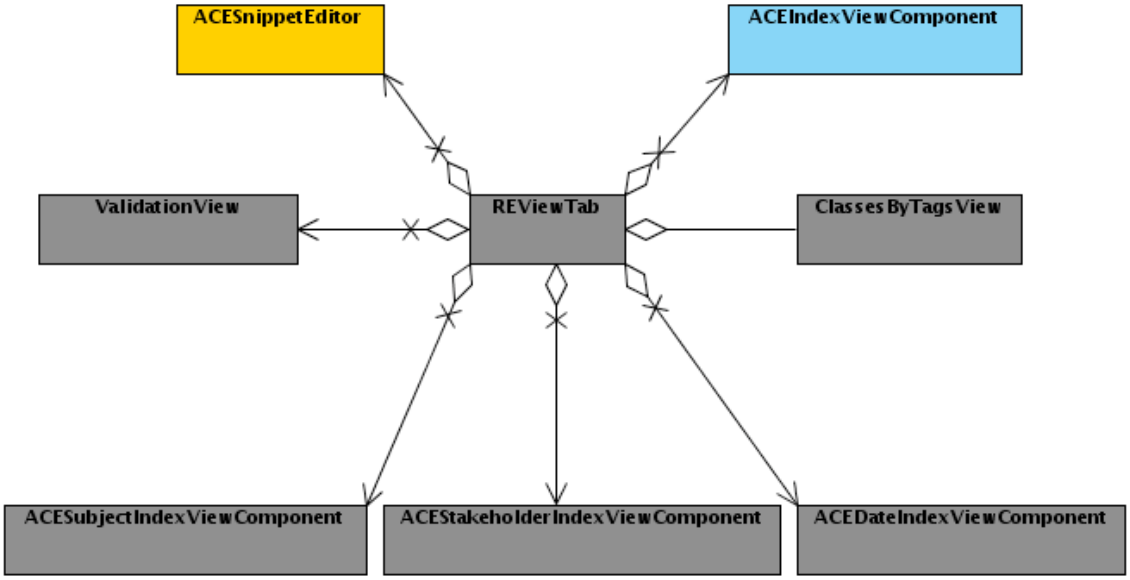

Figure 3.9: ACEView new Views

of all given sentences (a); edition fields to modify the sentences and a set of components that helps the user to inspect each sentence (b), like the list of tags used to mark such sentence (c), for instance. Figure 3.10 shows such component;

2. `ACEIndexViewComponent`: this component groups the snippets by the entities cited on them. Figure 3.11 shows such component;

3. `ClassesByTagsView`: this view groups the ontology classes by which tags mark them; Figure 3.12 shows such component;

4. `ACESubjectIndexViewComponent`: this component groups the snippets by the subject tag which marks them. This component can be seen at Figure 3.13;

5. `ACEDateIndexViewComponent`: This component groups the snippets by the date tag which marks them. It can be seen at Figure 3.14;

6. `ACEStakeholderIndexViewComponent`: This component groups the snippets by the stakeholder tag which marks them; Such component can be seen at Figure 3.15;

7. `ValidationView`: this view provides the functionality to check the consistency of the underlying ontology and in case there are conflicting concepts it shows the snippets which contain them, classified by subject and date tags, as can be seen at Figure 3.16;

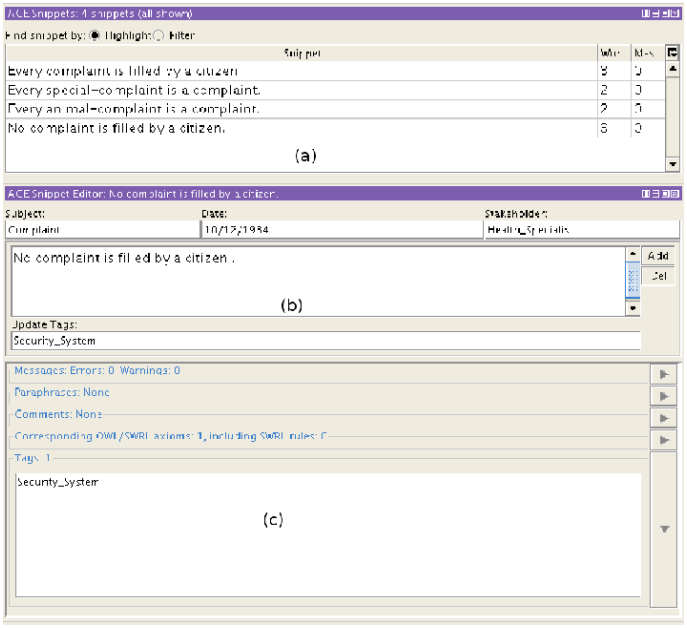

Figure 3.10: ACEView's ACESnippetEditor

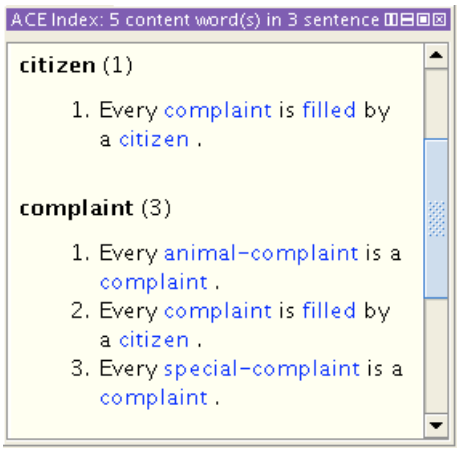

Figure 3.11: ACEView's ACEIndexViewComponent

Figure 3.12: ACEView's ClassesByTagsView

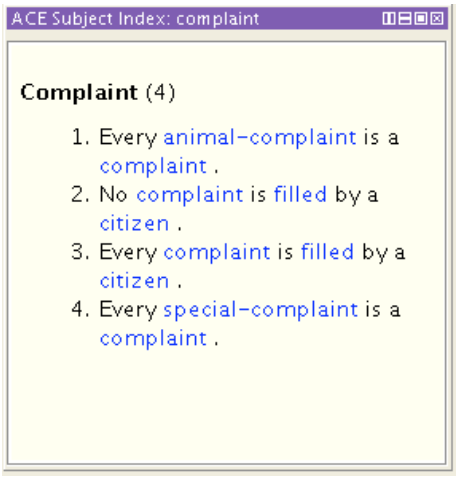

Figure 3.13: ACEView's ACESubjectIndexViewComponent

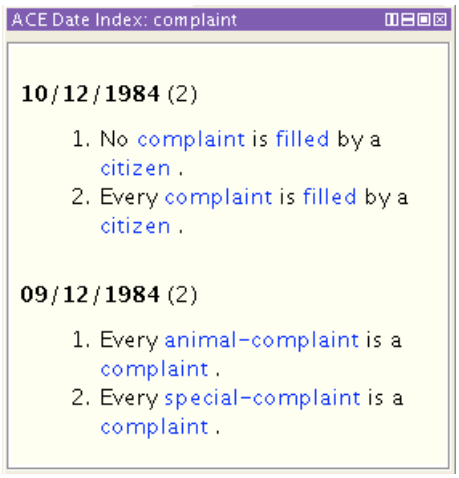

Figure 3.14: ACEView's ACEDateIndexViewComponent

Figure 3.15: ACEView's ACEStakeholderIndexViewComponent

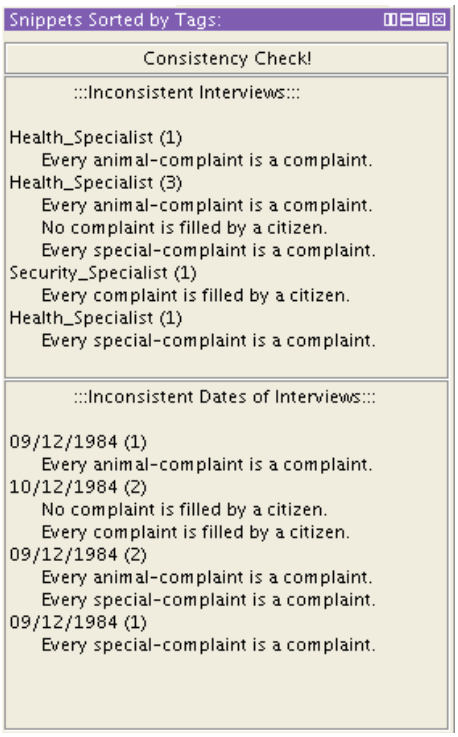

Figure 3.16: ACEView's ValidationView

3.3.1.2 Ontology package

This package is responsible for providing the infra-structure (i.e. metamodels and transformation files) needed to build a model representation of the domain ontology (output artifact activity 3a, 3.2). Such representation is necessary to implement the MDE transformation process executed in the PIM generation phase of our RE process (activity 6 of our process).

To formally define the ontology model representation, we have used an OWL metamodel, which describes the allowed elements of domain ontologies. We have chosen to use the *Ontology Definition Metamodel* - ODM [CRH+06], since it is specified by the OMG and it is considered a standard ontology metamodel.

At our implementation we made a slight modification at ODM, by introducing the *Tag* concept, which is responsible for keeping the information about the source of the provided knowledge (interviews), for traceability purposes. Such modification and the simplified metamodel can be seen in Figure 3.17. The metaclass `OntologyElement` was at the original version of ODM and was modified to add the tagging capability. Also, we introduced the `Tag` concept. The other metaclasses were reused from the ODM definition.

This package also provides a parser for reading the OWL files. We have implemented this parser using a set of ATL rules that are executed against the XML

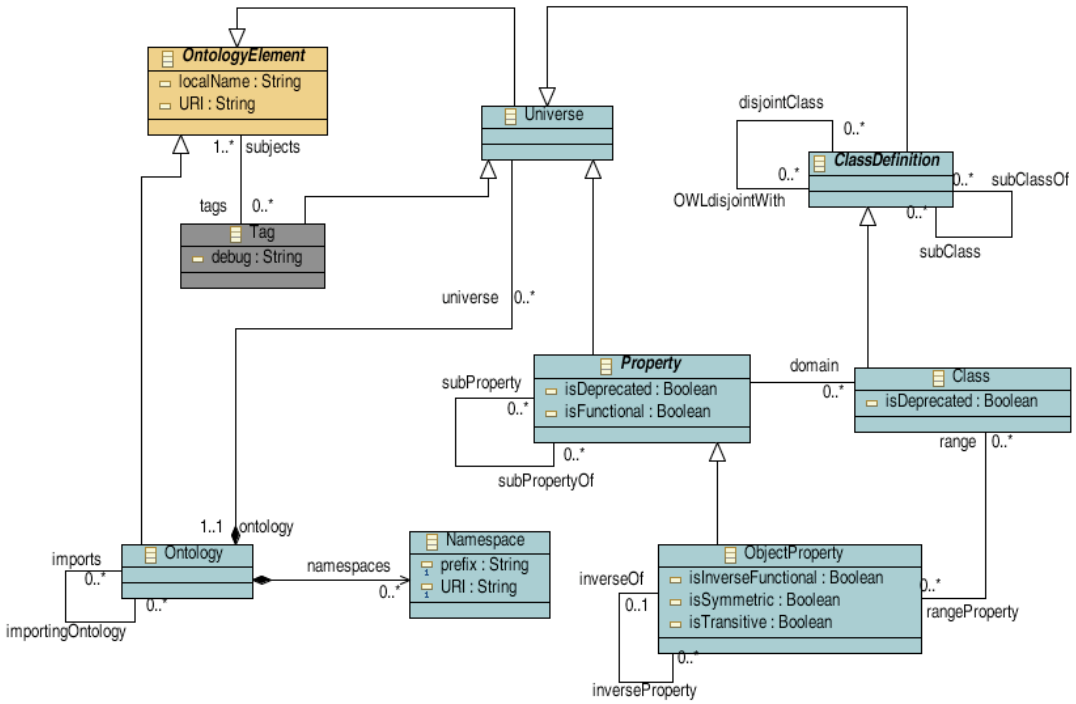

Figure 3.17: The modified Ontology Definition Metamodel

metamodel (provided by the XML package), which can be seen in Figure 3.18. These rules build an ODM model that is saved into a XMI file. We adopted the OWL ontology as the format accepted by our parser, since this representation is recommended by the W3C consortium. It is the reference specification used to specify OWL ontologies.

In order to construct the OWL parser we have implemented a library to process XML and OWL files. Such library provides common XML processing operations and OWL specific operations. These operations were designed to run against a DOM[5] XML processing model and are available as *ATL helpers* that can be used within any XML or OWL file. Following we will list the most important operations. We will use the ATL terminology to specify the parameters and return values types. The ATL rules format uses an exclamation point to separate the metamodel from its contained metaelement referred by the rule, i.e. METAMODEL!METAELEMENT.

- `getAttribute()`: this operation returns the value of the attribute given as a parameter or returns an empty *String* if this attribute does not exist;
- `getText()`: returns the text child activity belonging to the activity whose name was given as a parameter;

[5]http://www.w3.org/DOM

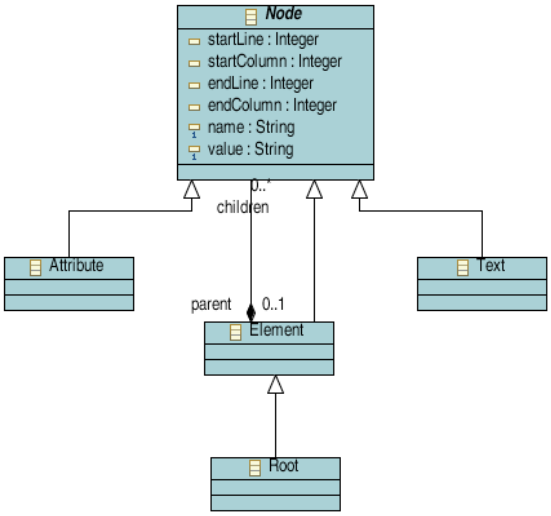

Figure 3.18: The XML Metamodel

- `getOWLClass()`: searches within the main ontology for the class whose name was given as a parameter and returns it or returns `OCL!Undefined` if such class does not exist;
- `isOwlClass()`: checks if the provided `XML!Element` represents an `OWL!Class`;
- `isOwlThing()`: checks if the provided `OWL!Class` is the *owl:Thing* named class, from which all `OWL!Class` descend;
- `getAllOwlClasses()`: returns all `OWL!Class`; and
- `getTag()`: searches within the provided ontology for the *OWL!Tag* element whose named was given as a parameter and returns it, or returns `OCL!Undefined` if such tag cannot be found.

The transformation file is composed of the following transformation rules:

- rule `Root2Ontology`: this rule builds an `OWL!Ontology` from the `XML!Root` element;
- rule `Class`: this rule builds `OWL!Class` elements from `XML!Element` instances, which represent elements of `OWL!Class` and are not *owl:Thing*;
- lazy rule `buildTag()`: this rule is called inside the `Class` rule after checking if already exists an `OWL!Tag` with the name caught at the owl annotation that references the `OWL!Class`, which is being built. If it is the case that such tag does not exist, this rule creates one and returns it; and

- rule `ObjectProperties`: this rule is responsible for creating `OWL!ObjectProperty` elements, which represent a property that connects two sets of concepts (i.e. `OWL!Class` elements) by means of the `domain` and `range` attributes.

3.3.1.3 ATL package

This package contains the core components responsible for executing the last phase of the proposed RE process, the PIM generation (activity 6, Figure 3.2). Such components are responsible for applying the MDE transformation rules to build UML class models from ODM models.

To do so, we have implemented a set of transformation rules in ATL to convert ODM entities and properties into UML class models:

- rule `Ontology2Package`: this rule transforms the main ontology into a root package that will contain all the elements in the UML model;
- rule `OWLClass2UMLClass`: this rule transforms an `OWL!Class` into an `UML!Class` keeping its properties and super-classes;
- rule `OWL2UMLProfile`: this rule creates a new profile for the main ontology in the ODM model. Such profile is used to group the stereotypes that identify which interview defined the entities used to built such class;
- lazy rule `Tag2Stereotype`: this rule is called to build a new `UML!Stereotype` from each tag. Then, the stereotype is applied to the `UML!Class` which corresponds to the source `OWL!Class`;
- rule `ObjectProperty2Property`: this rule transforms the `OWL!ObjectProperty` to an `UML!Property`;

3.3.1.4 UML and XML

These packages contain the UML and XML metamodels and are used along with the ODM metamodel to transform ODM into UML. The XML metamodel is used to build ODM models from XML files and was already presented in Figure 3.18.

We adopted OMG UML2 (v.2.1). Its metamodel was built using the EMF and it is available at the eclipse UML2 project[6]. Following we will present a diagram in Figure 3.19 that illustrates the UML metamodel main elements which were used in our transformation.

[6]http://www.eclipse.org/uml2

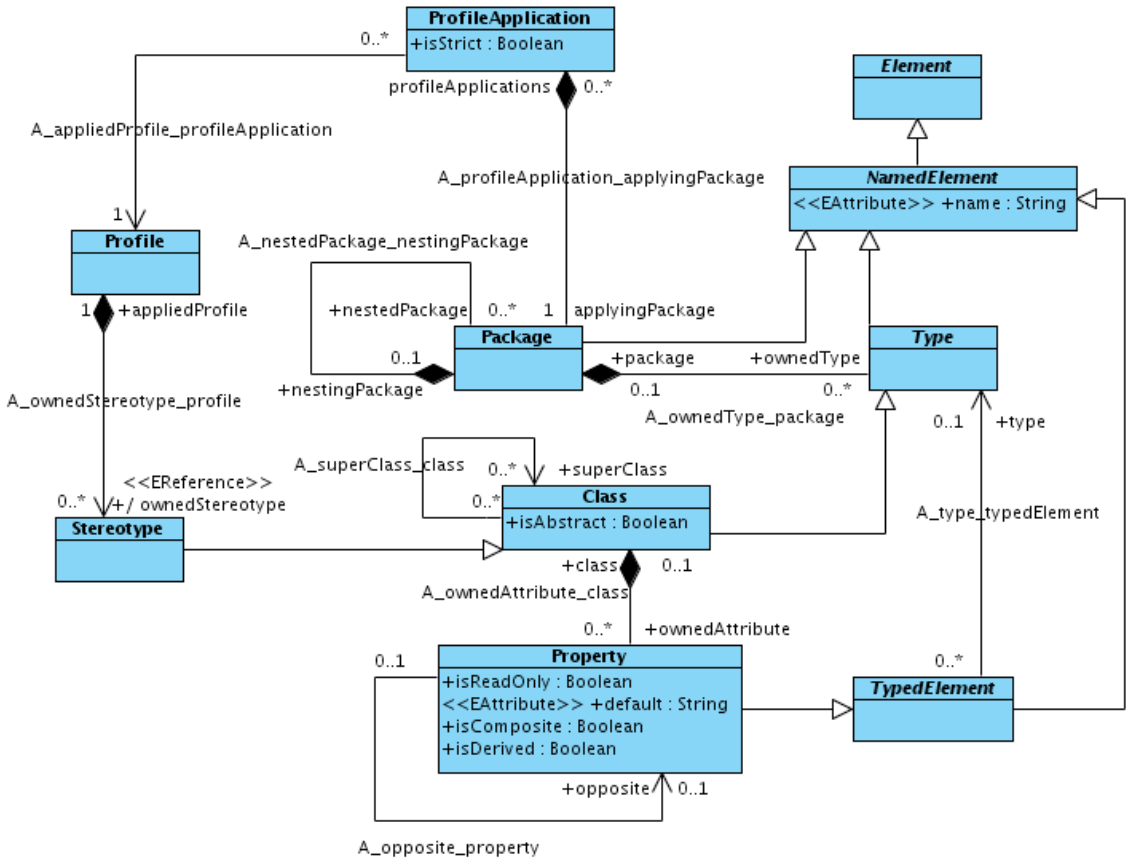

Figure 3.19: The UML Metamodel

Chapter 4

Case Study

We have conducted a case study to illustrate the execution of our process, showing its application and benefits. Our case study models the Health Watcher system (HW), a well-known testbed adopted by the AOSD Europe [1], whose goal is to register complaints about the public health system and also be used as guide to the public health care system, describing health units, and their specialties.

The system registers three kinds of complaints: *animal-complaint*, *special-complaint* and *food-complaint*. Each complaint has a set of data to identify the occurrence. The system also keeps track of each health unit and which diseases are treated by them.

The HW system is used to validate Aspect-Oriented projects and initiatives and we have chosen it because of the availability of documents describing its requirements and use cases. We have extended its specification to describe requirements related to the security aspects. Also, we have added information about the pharmacy, medical procedure and health unit staff (doctors, nurses, etc.) to the systems specification. After this extension, it has an appropriate domain size and the newer use cases specifications can show all the benefits of our process.

The complete description of the system can be found at [SLB02]. The system has several specification documents and 7 implementations. Each one adds new aspects to the system. All 7 versions share the same domain. We have compared this domain with the PIM generated by our tool. In order to obtain the domain we used the source code and a reverse engineering tool. We have compared the concepts one by one and such comparison showed that our process was able to capture all the concepts described at the previous domain version. From now on, we will refer to the domain obtained from the reverse engineering process as "Old Domain".

[1] http://www.aosd-europe.net

We adopted a *Goal/Question/Metrics* (GQM) approach [BCR94] to analyze the case study. Such analysis evaluates the process' effectiveness while dealing with the remaining open issues explained in Section 1. So, our evaluation provides a quantitative study divided according to the following RE open issues presente in section 1:

1. **Scope Definition** issues;
2. **Communication** issues;
3. **Knowledge Validation** issues; and
4. **Traceability Scheme** issues.

4.1 The Goal/Question/Metrics plan

The GQM approach is based upon the assumption that to measure software aspects in a purposeful way, an organization have to define the goals for itself or its processes; then it should map those goals to the data measures that intend to define them at the operational level, and provide a framework to interpret the collected data with respect to the stated goals [BCR94].

GQM is a top-down approach to establish a goal-driven measurement system for software development. The approach is divided into three layers: the organizational goals which defines measurement goals (conceptual level); the questions to address the goals (operational level); and the metrics that provide answers to the questions (quantitative level). Figure 4.1 illustrates the GQM paradigm and its division in levels as it was proposed by its authors [BCR94].

The usage of the GQM started with the process of evaluating defects of a set of projects at the NASA Space Flight Center [BW84]. Although the process was first designed to be used in order to evaluate particular goals at an specific project context, it has been expanded to broader contexts and goals [BCR94]. It is a widespread method to gather reliable empirical data and knowledge about an organization's software practices to drive systematic process improvements.

GQM is particularly useful for the following purposes [Wal05]:

- Understanding and baselining software practices of an organization;
- Guiding and monitoring software processes;

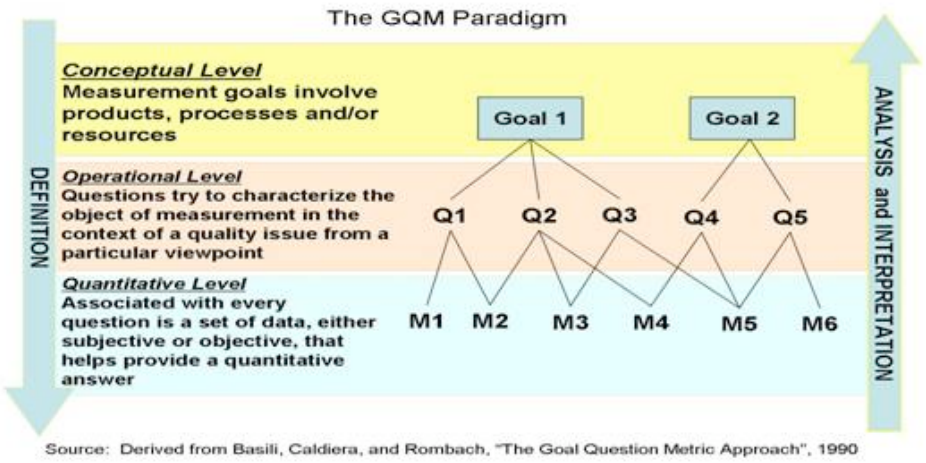

Figure 4.1: The GQM paradigm level division [BCR94]

- Assessing new software engineering technologies; and
- Evaluating and certifying improvement activities.

The result of the application of the GQM approach is the specification of a measuring system targeting a particular set of issues and a set of rules for the interpretation of the collected measurement data [BCR94].

Basically, the literature [BCR94, Wal05, BW84] describes the GQM as a six step process. The first three steps are used to identify the right metrics by means of measurement goals, whose definition is guided by the organization business goals. The last three steps define ways to collect the measurement data and use it do drive decision making and process improvements [Wal05] Following we will list the GQM six-step process [Bas05]:

1. **Develop** a set of corporate, division and project **business goals and associated measurement goals** for productivity and quality;
2. **Generate questions** that define those goals as completely as possible in a quantifiable way;
3. **Specify** the **measures** needed to be collected to answer those questions and track process and product conformance to the goals;
4. **Develop mechanisms** for data collection;
5. **Collect**, **validate** and **analyze** the **data in real time** to provide feedback to projects for corrective action; and

6. **Analyze** the **data** in a **postmortem** fashion to assess conformance to the goals and to make recommendations for future improvements.

GQM process starts with the definition of some measurement goals (conceptual level in Figure 4.1). Such definition is guided by the organization business process goals. After the definition of the measurement goals, the process team (project managers, development team, process analyst etc.) should define a set of questions (operational level in Figure 4.1) to further clarify and refine the goals. Such questions are used to describe the goals in a quantitative way [Bas05].

After the definition of the questions, the process team establishes metrics that will provide answers to the questions (quantitative level in Figure 4.1). What distinguishes GQM from other measurement paradigms [PKCS95, KN05, PWCC95] is its hierarchical tree structure used to maintain the relationships among goals, questions and metrics as one can see in Figure 4.1 [Wal05].

After the establishment of the appropriate metrics to answer the posed questions, the last three steps of the GQM process are concerned with how the metrics are going to be used in a way that ensures that the goals are going to be acquired. Most of the works in GQM [Bas05, BCR94, BW84, Wal05] discuss the importance of defining how the data are going to be collected and, more important, how the collected data should be organized and presented in order to maximize their value to the process team that will interpret the results in relation to the goals. The data measurement is very important and the bad use of it is described as the primary cause of failure in metric based process improvement and analysis programs [Wal05].

We followed the six-step GQM process proposed by [Bas05] in order to define our process improvement and analysis program to analyze and evaluate the effectiveness of the process that we proposed. Also the same program was used to analyze and evaluate the effectiveness of the tool that we've designed and implemented.

The first activity of the GQM process intends to define the measurement goals according to some business goals. Our main business goal is to prove that our process is effective while dealing with RE open issues (Section 1). So, we have defined four main measurement goals for our GQM plan and they were used to indicate that our process and tool are effective while dealing with RE open issues presented in Section 1.

Table 4.1, describes our **measurement goals** in order to evaluate our proposed process and tool. The descriptions of the measurement goals were written according to GQM template in [Wal05]. The data will be collected from the case study ex-

plained at the very beginning of this Section. The details related to the case study and process execution can be found at Section 4.3.

Table 4.1: The GQM Measurement Goals

GQM Measurement Goals	
First Goal	To **Analyze** the usage of the proposed process and tool in a requirements engineering phase of a software process; **For the purpose of** evaluating their usage effectiveness in real software development; **With respect to** dealing with the scope definition issues presented in Section 1; **From the viewpoint of** requirements engineer and clients stakeholders; **In the context of** information system development process.
Second Goal	To **Analyze** the usage of the proposed process and tool in a requirements engineering phase of a software process; **For the purpose of** evaluating their usage effectiveness in real software development; **With respect to** dealing with communication issues (Section 1) among teams involved in the software development process; **From the viewpoint of** requirements engineer and the software design team; **In the context of** information system development process.
Third Goal	To **Analyze** the usage of the proposed process and tool in a requirements engineering phase of a software process; **For the purpose of** evaluating their usage effectiveness in real software development; **With respect to** dealing with requirements validation and evolution issues (Section 1); **From the viewpoint of** requirements engineer, the software design team and client's stakeholders; **In the context of** information system development process.

Fourth Goal	To **Analyze** the usage of the proposed process and tool in a requirements engineering phase of a software process; **For the purpose of** evaluating their usage effectiveness in real software development; **With respect to** dealing with traceability issues (Section 1); **From the viewpoint of** requirements engineer and the software design team; **In the context of** information system development process.

It is worth noting that the evaluation of the case study is not concerned with measuring the productive gain and/or the quality improvement obtained while using the RE process and tool.

Following, at Tables 4.8, 4.9, 4.10 and 4.11 we will present a set of questions and measures that describes the goals defined to indicate the process effectiveness against each one of the RE process open issues presented in Section 1.

Table 4.2: GQM for Scope Definition Issues

Scope Definition Issues		
Question	**Metrics**	**Observations**
What is the difference between the number of entities at the HW domain generated using our process and the old domain, i.e. previous AOSD implementations ($E_{new-old}$)?	Ne_{new} = Number of Entities at the Domain Ontology.	This question intends to compare the data between the new version of the HW and the older one in order to characterize the process and tool's capability of discovering entities at textual descriptions.
	Ne_{old} = Number of Entities at the Domain of the HW previous version.	
	$E_{new-old} = Ne_{new} - Ne_{old}$	

What is the average number of entities per interview (N_{epi})?	N_e = Number of entities at the domain ontology. A_i = Total Amount of interviews with client's stakeholders. $N_{epi} = \frac{N_e}{A_i}$	This question intends to characterize the process' capability of isolating the entities provided at each interview. This number can also be used to evaluate the interview process.
What is the highest number of entities per subject tag (N_{epsu})?	N_{ej} = Number of entities at the domain ontology marked with the jth subject tag. A_s = Total Amount of subject in interviews with the client's stakeholders. $N_{epsu} = Max(\frac{N_ej}{A_s})$, being $j = 1...n$ where n is the last subject tag.	This question intends to characterize the process' capability of isolating the entities related to each subject. This number can be used to evaluate the complexity (based on the number of entities) of each part of the system concerned with a certain subject. These numbers are compared among them and the highest one indicates the subject with the highest number of entities (and the highest complexity)
What is The number of the inferred relationships (N_{infe})?	N_{infe} = Number of entities moved after the taxonomic classification.	This questions helps describing the process capability of discovering relationships between the entities.

Which stakeholder was involved at the larger number of conflicts (S_c)?	A_s = Total amount of sentences. S_{ci} = Total amount of conflicting sentences given by the i^{th} stakeholder. $S_c = Max(\frac{S_{ci}}{A_s})$, where $i = 1...n$, being n the total amount of stakeholders.	This question intends to characterize the ability of the process of discovering which stakeholders make a high number of mistakes in his/her specifications.
What is the average number of sentences removed after the validation activity (N_{aer})?	N_s = Number of sentences at the client's interviews. A_{rs} = Total Amount of Sentences removed after the validation activity. $N_{aer} = \frac{N_s}{A_{rs}}$	This question intends to evaluate the applied interview techniques. A low number represents a high number of inconsistent sentences at the interviews.

Table 4.3: GQM for Communication Issues

Communication Issues		
Question	**Metrics**	**Observations**
What is the amount of the removed entities at the PIM generation Phase ($E_{pim-domain}$)?	Ne_{domain} = Number of Entities at the Domain Ontology. Ne_{pim} = Number of Entities after the PIM generation.	This question helps to characterize the amount of data mistakenly captured as entities instead of primitive properties.

	$E_{pim-domain} = Ne_{pim} - Ne_{domain}$	
What is the average number of PIM classes per domain ontology entities (N_{cpe})?	N_c = Number of PIM classes. N_e = Number of entities at the domain ontology. $N_{cpe} = \frac{N_c}{N_e}$	This question intends to characterize the process' capability of mapping discovered entities into PIM classes.
What is the average number of inconsistent entities per number of removed sentences to fix such inconsistencies (N_{iprs})?	N_{ic} = Number of inconsistent concepts at the domain ontology. N_{rs} = Number of removed sentences to solve the ontology's inconsistencies. $N_{iprs} = \frac{N_{ic}}{N_{rs}}$	This question intends to characterize the amount of inconsistent knowledge at the requirements base. A high number means that very little work was done at the system's descriptions in order to solve the requirements knowledge inconsistencies.
What is The number of the inferred relationships (N_{infe})?	N_{infe} = Number of entities moved after the taxonomic classification.	This questions helps describing the process/tool capability of discovering relationships between the entities. Relationships that, otherwise, would remain undiscovered thanks to communication problems.

Table 4.4: GQM for Requirements Validation Issues

Requirements Validation Issues		
Question	**Metrics**	**Observations**
Which stakeholder was involved in the highest number of conflicts (S_c)?	A_s = Total amount Number of sentences.	This question intends to characterize the ability of the process of discovering which stakeholder has made more mistakes in his/her descriptions. It can be used to classify/identify irrelevant stakeholders to the software building process.
	S_{ci} = Total amount of conflicting sentences given by the i^{th} stakeholder.	
	$S_c = Max(\frac{S_{ci}}{A_s})$, where $i = 1...n$, being n the total amount of stakeholders.	
Which subject had the larger number of conflicts (Sub_c)?	A_s = Amount of sentences.	This question intends to characterize the process' ability of discovering which subject has the greatest amount of inconsistencies. Such data can be used to evaluate the complexity of relationships between entities at each subject. It can be used to classify and group the requirements by their subjects and level of difficulty to specify.
	S_{ci} = Total amount of conflicting sentences at the i^{th} subject.	
	$Sub_c = Max(\frac{S_{ci}}{A_s})$, where $i = 1...n$, being n the total amount of subjects and S_{ci} the amount of conflicts at subject i.	
What is the percentage of inconsistencies between facts about individuals and concepts(N_{iapis})?	N_{icABox} = The amount of the domain ontology concepts involved in ABox conflicts.	This question intends to characterize the process' ability of discovering inconsistencies between individuals and concepts.

	N_{ic} = The amount of conflicting individuals at domain ontology.	
	N_{cas} = The amount of conflicting sentences about facts whose individuals are involved at conflicts.	
	A_{cs} = The total amount of conflicting sentences.	
	N_{iapis} = $(\frac{N_{cas}}{A_{cs}}) * 100\%$	
What is the percentage of inconsistencies between concepts(N_{itpis})?	N_{icTBox} = The amount of concepts at the domain ontology involved in TBox conflicts.	This question intends to characterize the process' ability of discovering inconsistencies between concepts.
	N_{cts} = The amount of conflicting sentences about concepts that are involved at conflicts.	
	A_{cs} = The total amount of conflicting sentences.	
	N_{itpis} = $(\frac{N_{cts}}{A_{cs}}) * 100\%$	

Table 4.5: GQM for Traceability Issues

Traceability Issues		
Question	**Metrics**	**Observations**
What is the difference between the number of interviews conducted with the client's stakeholder and the number of SDS (Subject-Date-Stakeholder) tags at the domain ontology ($Ni_{Cli-Ont}$)?	Ni_{Cli} = The amount of interviews given by the client's stakeholders.	This question intends to characterize the process' ability of keeping metadata about the RE process after the construction of the domain ontology. This number is greater or equal to zero, where zero indicates that all interviews were kept and a non-zero value indicates that some interview information went missing.
	Ni_{Ont} = The amount of interview tags at the domain ontology	
	$Ni_{Cli-Ont} = Ni_{Cli} - Ni_{Ont}$	
What is the difference between the number of interviews data (Subject-Data-Stakeholder) before, at the domain ontology and after the PIM generation ($Ni_{Ont-PIM}$)?	Ni_{Ont} = The amount of interviews given by the client's stakeholders represented as SDS tags at the domain ontology.	This question intends to characterize the process' ability of keeping metadata about the RE process after the PIM generation. This number is greater or equal to zero, where a zero result indicates that all interviews were kept and a non-zero value indicates that some interview information went missing.
	Ni_{PIM} = The amount of interviews after the PIM generation.	
	$Ni_{Ont-PIM} = Ni_{Ont} - Ni_{PIM}$	

What is the difference between the number of concepts inside each interview (entities marked with the same SDS tags) before, at the domain ontology and after the PIM generation ($Nci_{Ont-PIM}$)?	$NciOnt_j$ = The amount of concepts defined at a j^{th} interview given by the client's stakeholders.	This question intends to characterize the process' ability of keeping the concepts untouched during the PIM generation. This number is greater or equal to zero, where a zero result indicates that all interviews were kept and a non-zero value indicates that some interview information went missing.
	$NciPIM_j$ = The amount of concepts (classes) inside the j^{th} interview at the PIM built from the domain ontology.	
	$Nci_{Ont-PIM} = \sum_{j=1}^{n} NciOnt_j - NciPIM_j$, where n is the total amount of interviews.	

4.2 The Case Study Scenario

In the case study we defined one scenario where the requirements engineer interviewed three different stakeholders: one responsible for non-functional requirements of system access control, identified by the name of **Security Specialist**. The information from this specialist was added to enhance the domain of the HW in order to show all the benefits of our approach. Another person is responsible for the whole complaining system, identified as **Health System Specialist**. The last one was responsible for the health service, identified as **Medical Specialist**. Table 4.6 lists the interview date, subject and stakeholder.

Table 4.6: Interview Data

Interview data		
Stakeholder	**Date**	**Subject**
Medical Specialist	*09/12/2008*	*Health System Organization*

Health System Specialist	*10/12/2008*	*Complaint System.*
Security Specialist	*11/12/2008*	*User Access Control*

During the first phase of the proposed process (requirements elicitation and analysis), the requirements engineer conducted interviews with the stakeholders and business analysts to gather information about the business domain. The Appendix A presents the detailed content of the interviews, divided by the interview tags (*Subject-Date-Stakeholder*).

The following subsection will explain the process execution after the first process activity (Figure 3.2, activity 1a and 1b), where the RE engineer has to gather information from client's stakeholders. At our example, we will suppose that the engineer collected information only from interviews. We did not use documents describing the domain because such documents would require translations to English. Furthermore, the definition of document revision techniques are out of the scope of this work.

4.3 The Process Execution

The second activity (activity 2, Figure 3.2) of the first phase is also done by the requirements engineer, whose job is to collect all the relevant information about the domain of the system and feed the NCL editor with it. The result of this activity is the description of the system containing only the relevant information. The requirements engineer should also register the information about what interview brought each part of knowledge up (client's view). Figure 4.2 illustrates the extended AceView after the execution of activity 2 of the proposed process. The Protégé view (a) shows the (ACE) sentences sorted by the Subject tags in which they have appeared. View (b) shows the sentences sorted by the date that they occurred. View (c) shows the other tags used to mark the selected sentence. Finally, view (d) shows all the sentences that contain each concept.

Figure 4.2 shows that the sentence "*Every employee has a password.*" is marked with three ordinary tags: "*Security*", "*Access*", and "*Control*". Moreover, such

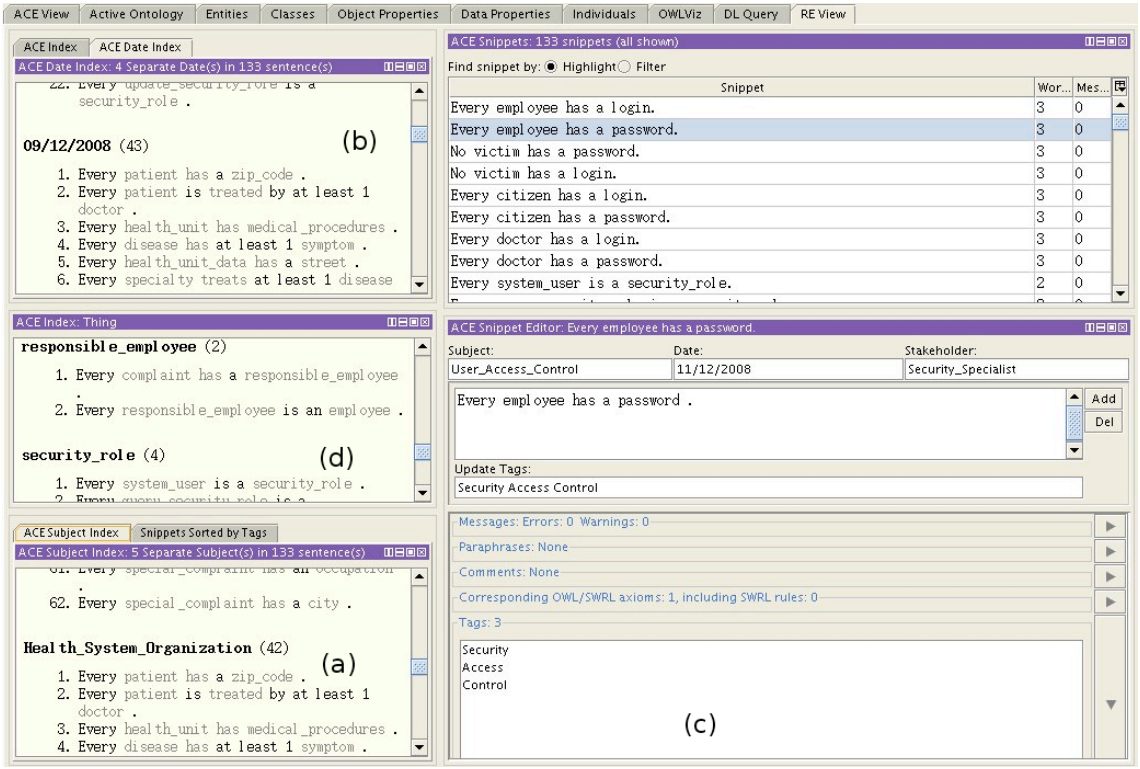

Figure 4.2: The plugin's new Views

sentence is also marked with the subject tag "*User_Access_Control*", the date tag "*11/12/2008*" and the stakeholder tag "*Security_Specialist*". So, from the SDS tags, one can conclude that the sentence was given by the security specialist, at December 11, 2008 and it is related to the subject user access control.

According to our proposed RE process, after feeding the tool with the collected requirements information, the next step is to build the domain ontology (activity 3a, Figure 3.2). Since AceView is a plug-in for the Protégé tool, the ontology, which represents the development team's view is built on-the-fly while sentences, collected by means of the interviews or document reviewing, are added into the tool. So, nothing needed to be done. Figures 4.3 and 4.4 show the generated ontology using a graph notation where the edges represent *is-a* relationships between the concepts, expressed as nodes. The graph was divided in two parts for readability's sake. The graph should be read from right to left and the relationship between the nodes are of the "is-a" type. Such relationship is denoted by the edges between the nodes. Take for instance the *Doctor* concept, it is-a *update security role*, which is-a *security role*, which is a *Thing*.

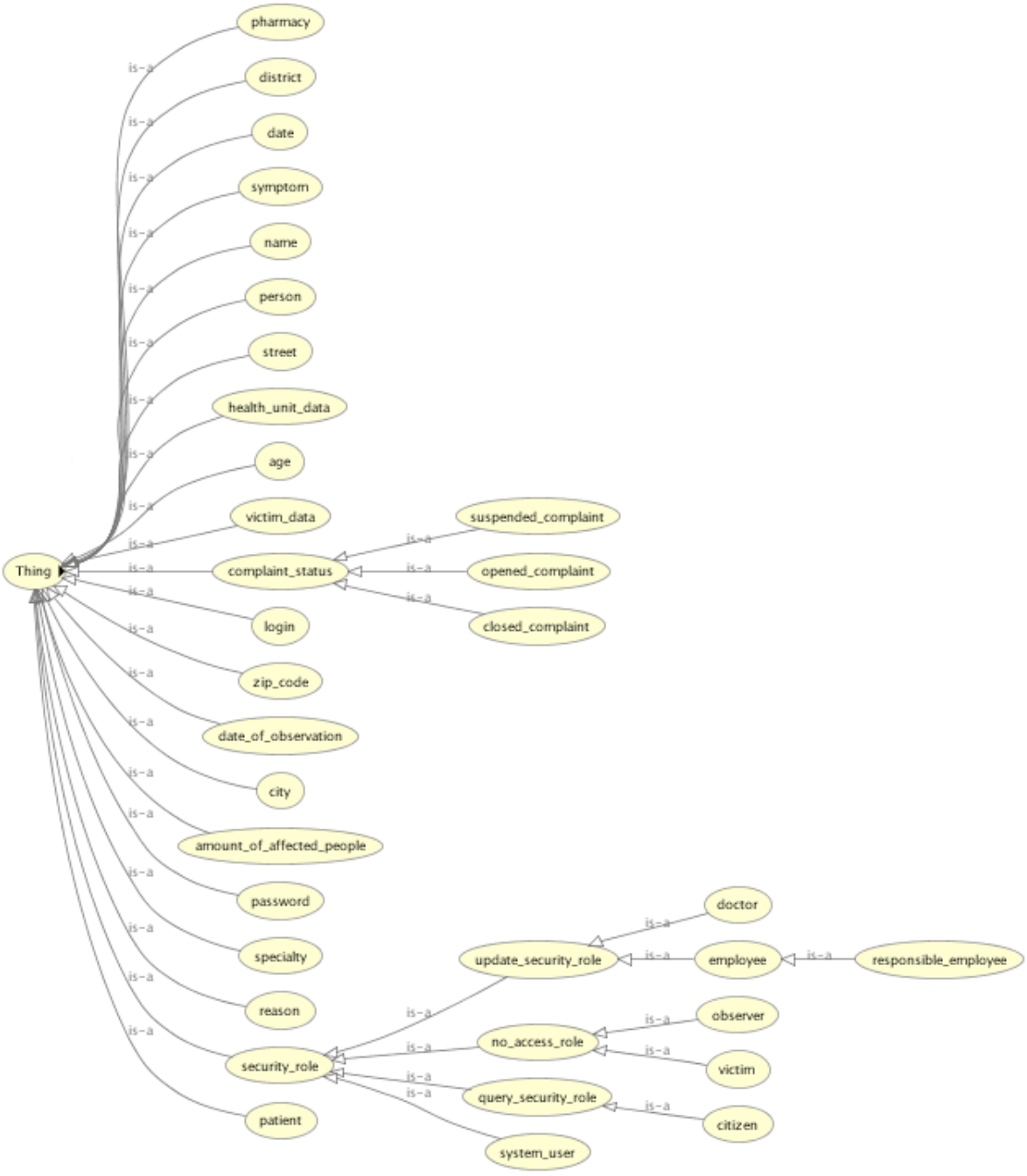

Figure 4.3: Built Domain Ontology part 1

Figure 4.4: Built Domain Ontology part 2

The next phase is the requirements validation where the ontology consistency is checked by a reasoner. In our tool, this was done by the Pellet reasoner bundled with the Protégé tool. Besides the domain ontology consistency, new relationships may be discovered through the taxonomic classification process. The following subsection will resume the inconsistencies discovered during the validation phase.

4.3.1 Inconsistencies and Taxonomic Classification

Our process, as already said, is able to discover both TBox and ABox inconsistencies. At our case study we were able to find ten TBox inconsistencies. Following we will enumerate each inconsistency and discuss the adopted strategy to solve them.

"*Every victim is a system_user.*" was the sentence responsible for the first detected inconsistency. The sentence was declared by the Health System Specialist at December 10, 2008 and it generated conflicts with three other sentences: "*No victim has a password.*" "*No victim has a login.*" and "*If something X is a system_user then X has a login and a password.*", all given by the Security Specialist at December 11, 2008. The conflict happens because the second stakeholder states that no victim has a password nor a login and that to be a system user, these data are needed. The first stakeholder contradicts such information by stating that every victim is a system user, even though it does not have a password nor a login.

In order to solve this conflict, the requirements engineer conducted a new interview with both conflicting stakeholders and resolved to remove the conflicting sentence "Every victim is a system_user.". Figure 4.5 shows the Validation View of our tool after the validation phase. It provided important information about the source of the conflicting sentences, showing which stakeholders are in conflict at the top frame, in this case the Health System and Security Specialists. Also, it provides information about the date that each conflicting sentence was given at the bottom frame.

Figure 4.6 shows the Protégé inferred hierarchy view that lists which concepts are in conflict along with all classes to which they are related. This view can be used together with the Validation and the ACE Index view that lists all sentences at which each concept was used, as can be seen in Figure 4.7. Crossing these three information facilitates the tracking of the problem's origin.

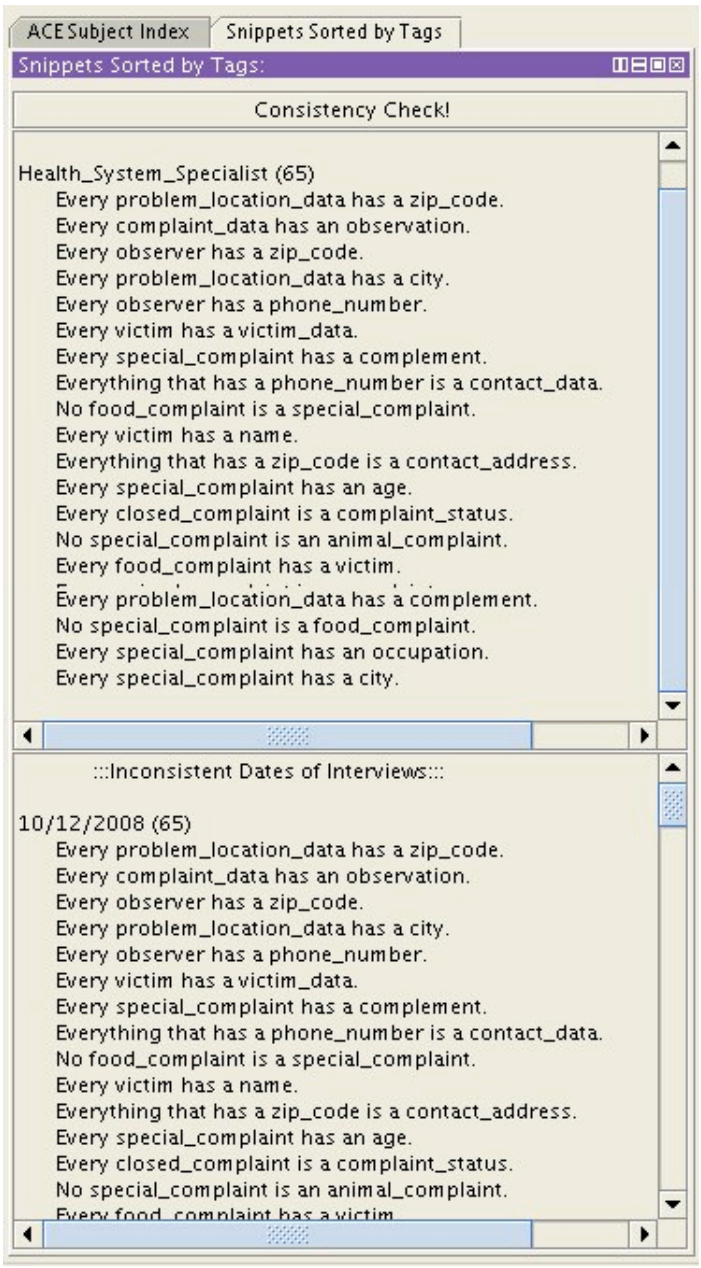

Figure 4.5: First Inconsistency shown at the Validation View

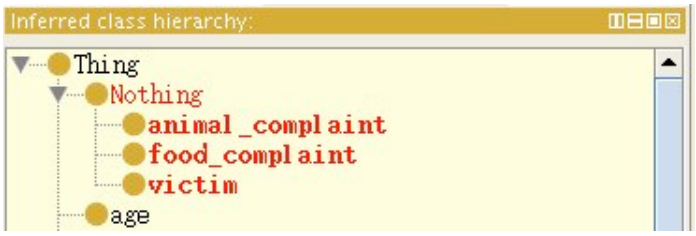

Figure 4.6: Conflicting concepts shown at the Protégé inferred hierarchy view

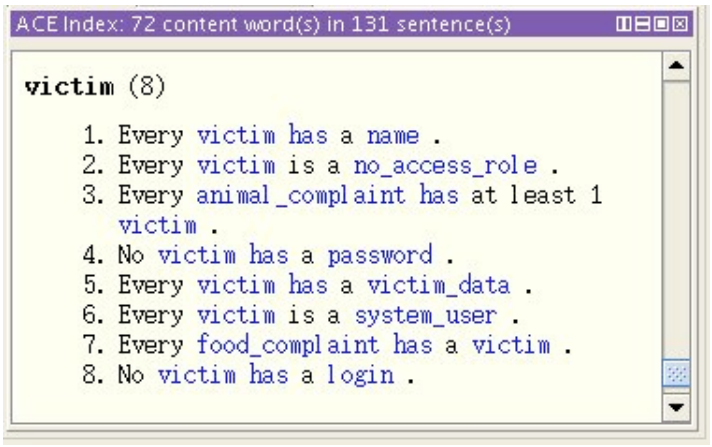

Figure 4.7: Sentences at which the conflicting concepts were used shown at the ACEIndex View

The second inconsistency was a result of the sentence "*Everything that has a problem_location_data is an animal_complaint.*" given by the Health System Specialist. It gets in conflict with the sentence "*Every special_complaint has a problem_location_data.*" also given by the same stakeholder, both given at December 10, 2008. The conflict was a result of the definition of animal complaint and special complaint as disjoint concepts. Such information was given by the sentence: "*No animal complaint is a special_complaint.*", also given by the same stakeholder at the same date.

In order to solve this inconsistency the RE engineer conducted a new interview with the Health System Specialist and he decided to remove the sentence "*Everything that has a problem_location_data is an animal_complaint*" since he though to be interesting that *special_complaints* also have *problem_location_data*. Figure shows the Protégé inferred hierarchy view, which shows the animal_complaint and special_complaints as conflicting concepts.

Figure 4.8: Conflicting concepts shown at the Protégé inferred hierarchy view

The third inconsistency was a result of the sentences "*Everything that executes medical_procedures is a doctor.*" and "*Every nurse executes at least 1 medical_procedure.*". This inconsistency is a result of the constraint that states that "*No doctor is a nurse.*" i.e. *doctor* and *nurse* are disjoint concepts - $doctor \cap nurse = \emptyset$. All sentences were given by the Medical specialist at December 9, 2008. Figure 4.9 shows the conflicting concepts related to *medical_procedure*. In order to solve such conflict, the RE engineer interviewed again the Medical Specialist and he recognized that both doctors and nurses execute *medical_procedure*, so the RE engineer removed the sentence "*Everything that executes medical_procedures is a doctor.*".

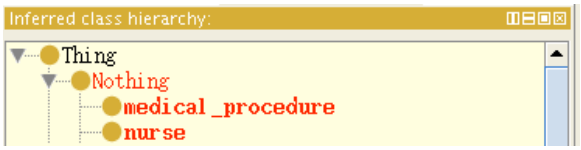

Figure 4.9: Conflicting concepts shown at the Protégé inferred hierarchy view

The fourth inconsistency was a result of the following conflicting sentences: "*Everything that has a symptom is a disease.*" given by the Medical Specialist at December 9, 2008, and "Every food_complaint has at least 1 symptom.", given by the Health System Specialist at December 10, 2008. These concept is a result of the complaint super class that states that "No complaint is a disease." i.e. $disease \cap complaint = \emptyset$. Figure 4.10 shows the conflicting concepts in Protégé inferred hierarchy view. This time we chose to use the ontology graph representation to show that it also can display inconsistent concepts, which are shown as red nodes.

Figure 4.10: Conflicting concepts shown at the Protégé graph view

In order to solve such conflict, the RE engineer conducted new interviews with both conflicting stakeholders and they come to the conclusion that a *food_complaint* should also have *symptoms*, this was done by loosing the restriction that states that "*Everything that has a symptom is a disease.*", so the interviewer changed such sentence to "*Every disease has at least 1 symptom.*".

The fifth inconsistency was a result of the the conflicting sentence given by the Security Specialist: "*Every nurse is a no_access_role.*" at December 11, 2008. This sentence conflicts with the sentences: "*Everything that has a login and a password is a system_user.*" and "*No system_user is a no_access_role.*" also given by the same stakeholder at the same date. The root of the conflict is the taxonomic process that classifies the nurse concept as a *system_user*, since it has a *login* and *password* associated. So, the *nurse* concept is, at the same time, a *system_user* and a *no_access_role*. Figure 4.11 shows the *nurse* concept represented as a red node because it is in conflict. The nurse concept does not have an *is-a* labeled arch connecting it with the *system_user* concept because it is also classified as as subclass of *Nothing*, as can be seen in Figure 4.12, so it can be a subclass of any concept. Thus, the graph representation after the taxonomic classification would show an arch connecting the nurse concept with every concept in the ontology.

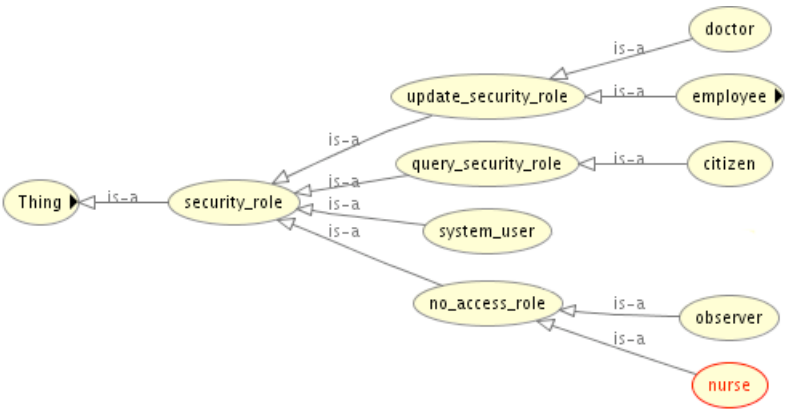

Figure 4.11: Conflicting concepts shown at the Protégé graph view

Figure 4.12: Conflicting concepts shown at the Protégé inferred hierarchy view

In order to solve these conflicts, the RE engineer conducts a new interview with the Security Specialist and he recognizes the mistake and he came to the conclusion that a *system_user* can also be a *no_access_role*. The RE engineer then removes the sentence that states that "*No system_user is a no_access_role.*".

The sixth inconsistency was a result of the following conflicting sentences: "*Every suspicious_food_establishment is a problem_location_data.*" and "*Everything that has a problem_location_data is an animal_complaint.*", both stated by the Health System Specialist at December 10, 2008. These sentences conflict due to the fact that *complaint* and *suspicious_meals* are disjoint sets i.e. $complaint \cap suspicious\_meal = \emptyset$.

In order to solve the sixth conflict, the RE engineer conducted a new interview with the Health System Specialist and he decided to remove the sentence "*Every suspicious_food_establishment is a problem_location_data.*" since it defines that a *suspicious_food_establishment* is a problem without any further analysis. Figure 4.13 shows the representation of such conflict using the Protégé graph view.

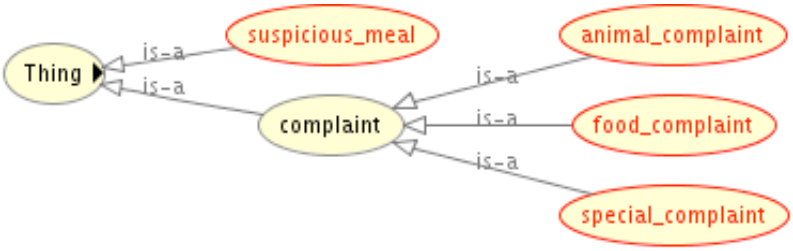

Figure 4.13: Conflicting concepts shown at the Protégé graph view

The seventh inconsistency was a result of the following sentences: "*Everything that has a victim is an animal_complaint.*" and "*Every food_complaint has a victim*", both given by the Health System Specialist at December 10, 2008. And the root of the conflict resides in the fact that *food_complaint* and *animal_complaint* are disjoint concepts i.e. $food\_complaint \cap animal\_complaint = \emptyset$. Such inconsistency also can be seen at Figure 4.13.

The eighth inconsistency was a result of the sentences: "*Everything that has a treatment is a disease.*" given by the Medical Specialist at December 9, 2008 and "*Every food_complaint has a treatment.*", given by the Health System Specialist at December 10, 2008 and it happened because of the same reason as the fourth inconsistency, the *food_complaint* concept is disjoint from the *disease* concept.

To solve this inconsistency the same strategy was adopted, the restriction that only diseases have treatment has been loosed and replaced by a new one that states that "*Every disease has at least 1 treatment.*". This inconsistency can be seen at Figure 4.14.

Figure 4.14: Conflicting concepts shown at the Protégé graph view

The ninth inconsistency was a result of the sentences: "*Everything that has a symptom is a disease.*" and "*No patient is a disease.*", both provided by the Medical Specialist at December 9, 2008. This case is similar to the fourth inconsistency case and is solved by the same modification, which looses the constraint stating that only diseases have symptoms.

Finally, the last inconsistency was a result of the sentences: "*No observer is a citizen.*" e "*Every observer registers at least 1 animal_complaint.*", both given by the Health System Specialist at December 9, 2008. The root of this conflict resides in the fact that an *animal_complaint* is a sub class of *complaint* and that only *citizens* can register *complaints*. Also, at the same time *citizen* and *observers* are disjoint concepts. The set of conflicting classes can be seen in Figure 4.15, using the Protégé graph view.

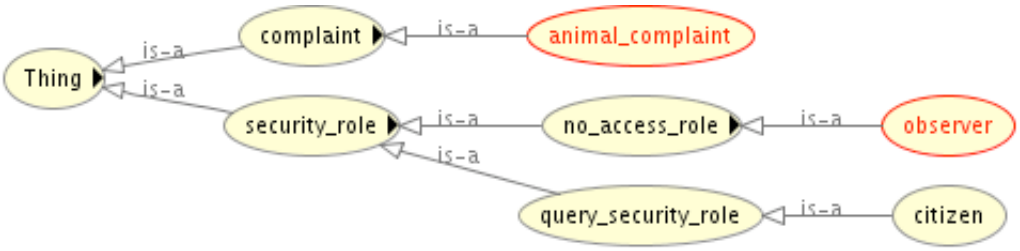

Figure 4.15: Conflicting concepts shown at the Protégé graph view

In order to solve this inconsistency all we had to do was to remove the *observer* and *citizen* disjointness sentence.

Our process has also identified ten ABox knowledge, i.e. the ontology individuals. They were all solved using the same strategy, removing the ABox sentences from the RE knowledge base in order to continue the reasoning process unlocking the knowledge base. Figure 4.16 shows the reasoning response while trying to process an inconsistent ontology.

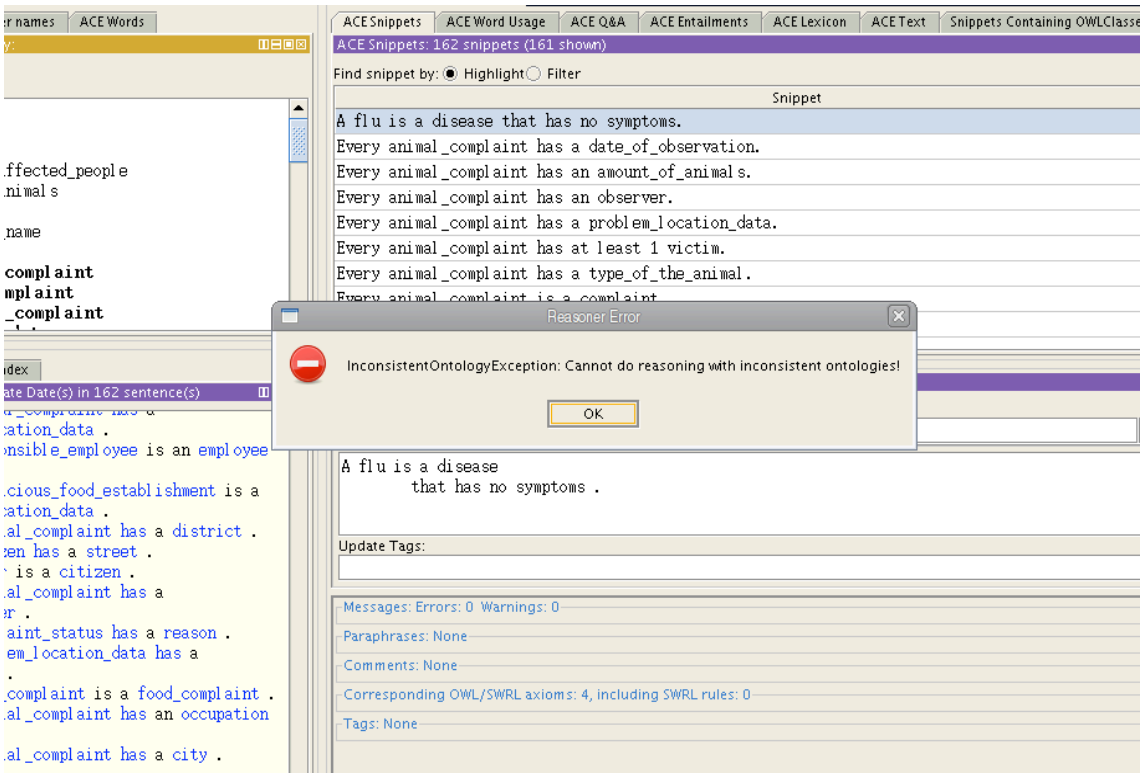

Figure 4.16: Reasoner response to an inconsistent knowledge base

Following, at Table 4.7 we will list the ABox sentences along with the TBox sentence with which it is in contradiction and the stakeholder who gave the ABox statement.

Table 4.7: ABox Inconsistencies

ABox Inconsistencies		
ABox Sentence	**TBox Sentences**	**Stakeholder**
"*A victim has a password.*"	"*No victim has a password.*"	*Health System Specialist*
"*A food_complaint has no victim.*"	"*Every food_complaint has at least 1 victim.*"	*Medical Specialist*
"*A nurse has no password.*"	"*Every nurse has a password.*"	*Health System Specialist*
"*A flu is a disease that has no symptoms.*"	"*Every disease has at least 1 symptom.*"	*Health System Specialist*
"*A visitor_doctor is a doctor that has no login.*"	"*Every doctor has a login.*"	*Medical Specialist*

"*A general_practitioner is a doctor that has no specialty.*"	"*Every doctor has at least 1 specialty.*"	*Medical Specialist*
"*An auxiliar_nurse is a nurse that executes no medical_procedure.*"	"*Every medical_procedure is executed by a nurse.*" and "*Every nurse executes at least 1 medical_procedure.*"	*Medical Specialist*
"*A natural_medicine is a drug that has no side_effects.*"	"*Every drug has at least 1 side_effect.*"	*Medical Specialist*
"*An observer registers a complaint.*"	"*If something X registers a complaint then X is a citizen.*" and "*No observer is a citizen.*"	*Health System Specialist*
"*An animal_poisoning_complaint is an animal_complaint.*" and "*An animal_poisoning_complaint is a food_complaint.*"	"*No food_complaint is an animal_complaint.*"	*Health System Specialist*

After solving the inconsistencies, the taxonomic classification takes place the new ontology built can be seen at Figures 4.17 and 4.18. We have chose to split the figure in two for readability's sake.

At our case study, thirty three new relationships were inferred by the reasoner. The Figure 4.3.1 shows the Protégé Inferred Axioms view. Each line at the view shows a *subClassOf* axiom, whose used to express the subclass relationship between two concepts.

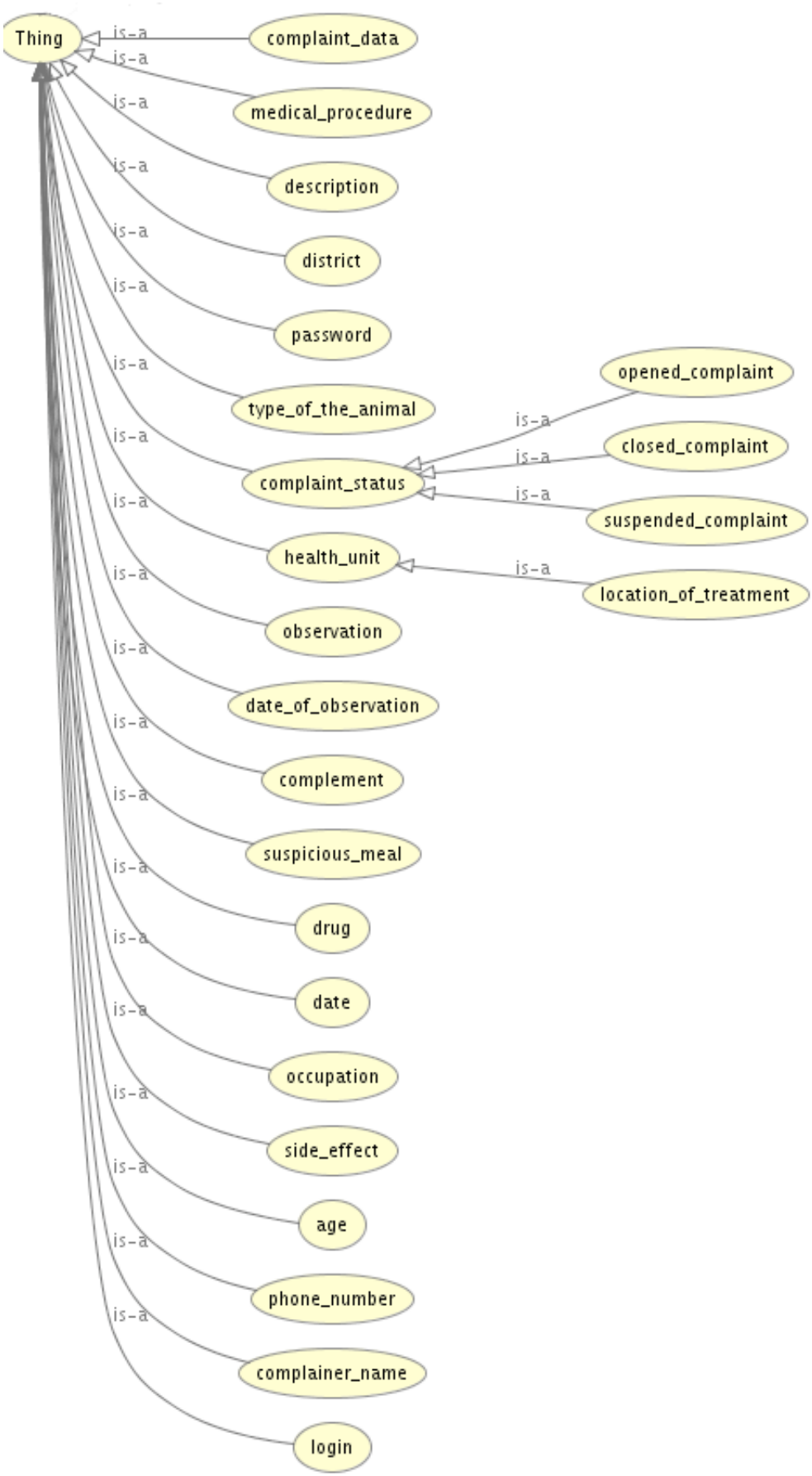

Figure 4.17: Inferred Hierarchy (a)

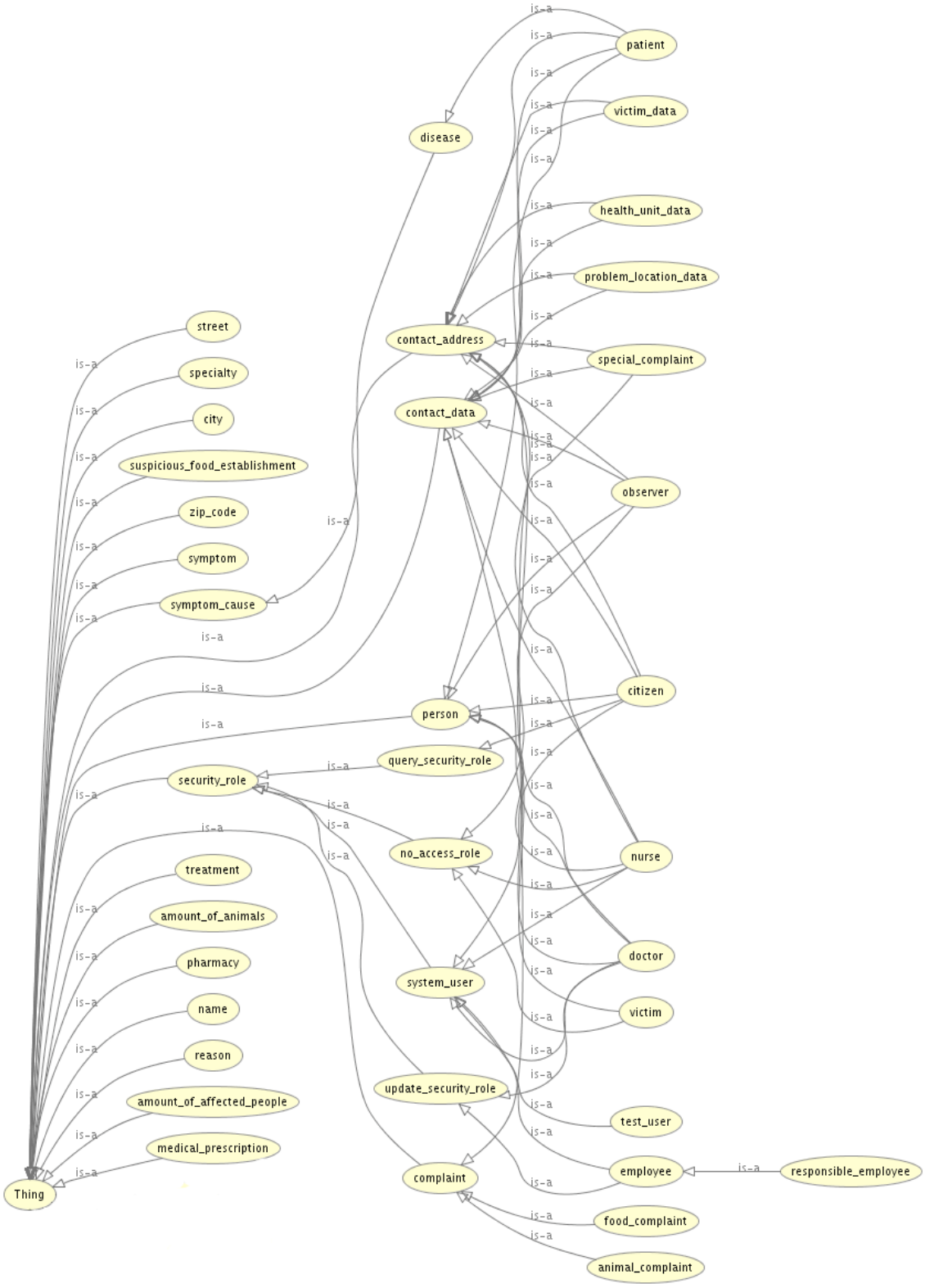

Figure 4.18: Inferred Hierarchy (b)

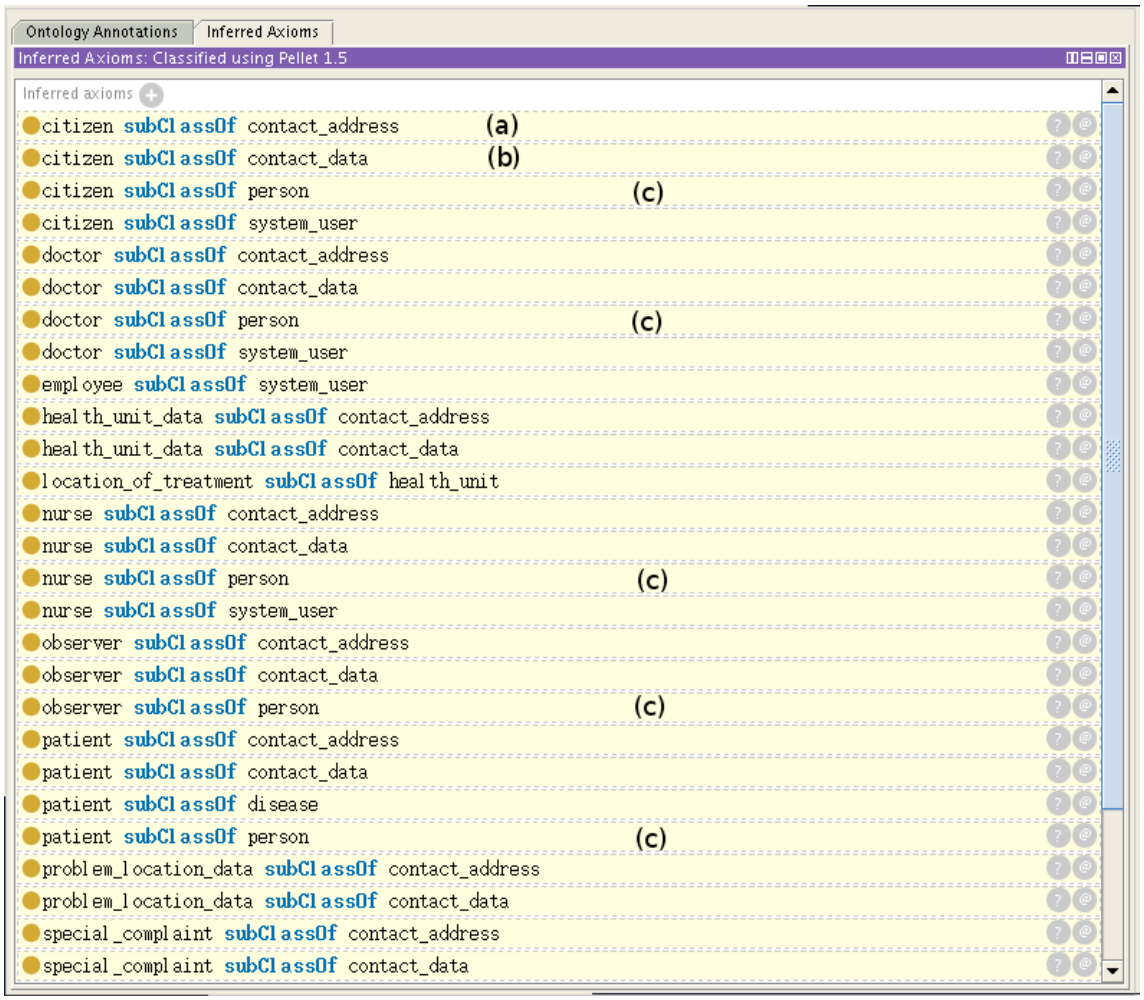

Figure 4.19: Protégé Inferred Axioms View

We would like to point some of the inferred relationships. The (a) axiom classifies a citizen as a *contact_data*, while the (b) axiom classifies it as a *contact_address*. The (c) axioms are a result of the sentence "*Everything that has a name is a person.*".

Following the process execution we've reached the PIM generation phase. The first activity of such phase is the domain cleaning activity. At our case study we have removed the properties which we thought to be primitive types, like: *name*, *phone_number*, *address*, etc. After the cleaning activity, it is time to generate the PIM, executing the Ant script with the defined transformation flow. We have divided the generated PIM according to their interviews for readability's sake. Figure 4.3.1, 4.3.1 and 4.3.1 shows, in order, the PIM for the Security Specialist, Health System Specialist and Medical Specialist interviews.

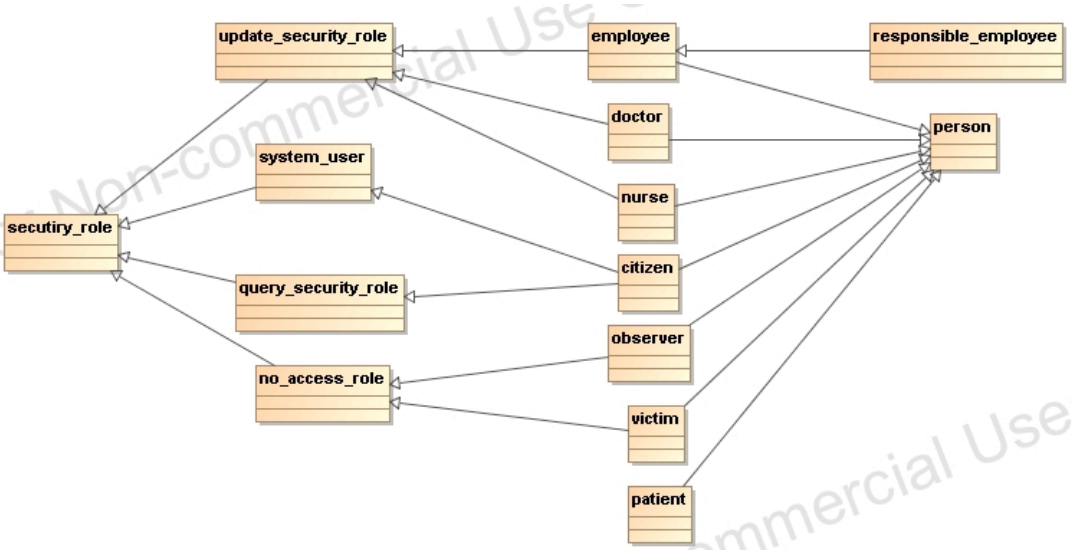

Figure 4.20: Generated PIM for the interview with the Security Specialist

4.4 The Case Study Evaluation

Following we will present the collected metrics values for our GQM questions and provide some interpretations for such values. We have structured this section according to the section 4.1 when these questions were first presented.

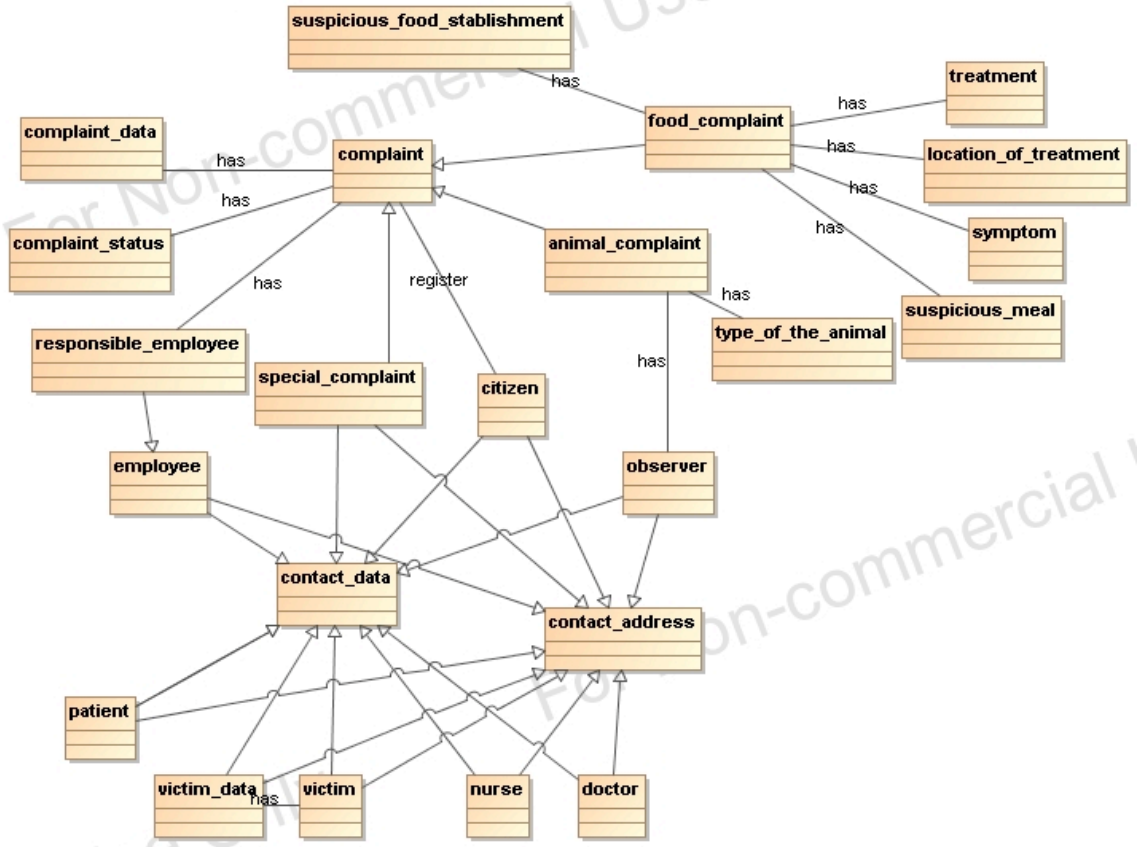

Figure 4.21: Generated PIM for the interview with the Health System Specialist

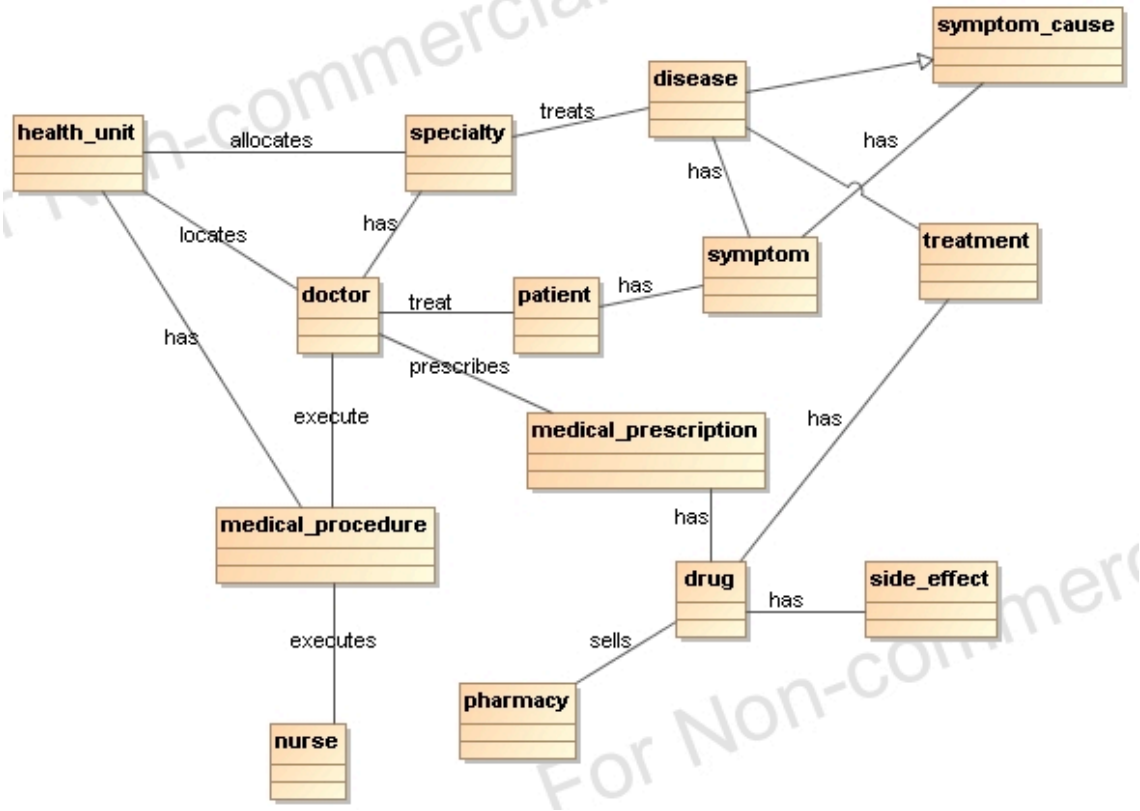

Figure 4.22: Generated PIM for the interview with the Medical Specialist

Table 4.8: GQM for Scope Definition Issues - Metrics values

Scope Definition Issues		
Question	**Metrics**	**Interpretations**
What is the difference between the number of entities at the HW domain generated using our process and the old domain, i.e. previous AOSD implementations($E_{new-old}$)?	Number of Entities at the Domain Ontology. $Ne_{new} = 25$	The 25 discovered concepts includes all the concepts specified at the HW implementations. So we were able to identify the same scope than the original HW version.[2]
	Number of Entities at the Domain of the HW previous version. $Ne_{old} = 10$	
	$E_{new-old} = 25 - 10 = 15$	
What is the average number of entities per interview (N_{epi})?	Number of entities at the domain ontology. $N_e = 64$	This number can be used to estimate the size of each "subsystem" whose information was provided by each stakeholder. Such information can be used to estimate the amount of resources to be allocated to each subsystem. The *Medical_Specialist* revealed the existence of 22 entities. The *Health_System_Specialist* 33. And the *Security_Specialist* 9.
	Total Amount of interviews with client's stakeholders. $A_i = 3$	
	$N_{epi} = \frac{N_e}{A_i} \simeq 21.33$	

[2]The comparison showed that our process captured a larger number of entities. The initial value of Ne_{new} was high (64) because our process extended the HW basic domain. Also, it did not take into account the primitive properties. So, primitive attributes are considered ontology concepts. After removing the concepts that represent primitive attributes and the new requirements related to medical organization of the health unit, the pharmacy and the security system, our Ne_{new} reduces to 25.

What is the highest number of entities per subject (N_{epsu})?	Number of entities at the domain ontology marked with the jth subject tag. $N_{e1} = 22$, $N_{e2} = 33$,$N_{e3} = 9$ Total Amount of subject in interviews with the client's stakeholders. $A_s = 3$ $N_{epsu} = Max(\frac{N_e j}{A_s}) = 11$, being $j = 1...n$ where n is the last subject tag.	This number can be used to evaluate the complexity of each part of the system concerned with a certain subject. The result of the *Complaint_System* as the highest number of entities (33) shows that this subject is an strong candidate for reviewing.
What is the number of entities removed from the generated PIM (N_{e-epim})?	Number of entities at the domain ontology. $N_e = 64$ Total Amount of entities at the generated PIM. $N_{epim} = 39$ $N_{e-epim} = N_e - N_{epim} = 25$	This number means that from 64 entities, the system engineer had to remove 25. It can be seen as the amount of work needed while "cleaning" the domain ontology. The closer to zero, the better. Our case study had a high number of entities to be removed. This happened because of the ACE NCL inability to represent primitive properties, like Strings, Integers, etc...

What is The number of the inferred relationships (N_{infe})?	Number of entities moved after the taxonomic classification. $N_{infe} = 17$	This number shows that our process did find a few new relationships at the domain ontology, for instance the definition of the *System_User* concept and also the *person* concept and the *Contact_data* concept. The greater the distance from zero, the better, it shows that a lot of relationships, that otherwise would remain undiscovered were found.
Which stakeholder was involved at the larger number of conflicts (S_c)?	Total amount of sentences. $A_s = 148$	This number shows that the system presented a low level of inconsistency even considering the stakeholder that has gave the largest number of inconsistent statements. At our example, the stakeholder involved at the larger number of conflicts was the $S_{Health\_System\_Specialist}$ with13 inconsistences.
	S_{ci} = Total amount of conflicting sentences given by the i^{th} stakeholder. $S_{Health\_System\_Specialist} = 13$(7 TBox, 6 ABox), $S_{Medical\_Specialist} = 8$ (2 TBox, 6 ABox), $S_{Security\_Specialist} = 1$ (TBox only)	
	$S_c = Max(\frac{S_{ci}}{A_s}) \simeq 0.09$	

Table 4.9: GQM for Communication Issues

Communication Issues		
Question	**Metrics**	**Observations**

What is the average number of sentences removed after the validation activity (N_{aer})?	Number of sentences at the client's interviews. $N_s = 148$	This number indicates that almost 1 sentence is invalid at every 10 Sentences. The lower the coefficient the better. We believe that the communication has flown at an acceptable level.
	Total Amount of Sentences removed after the validation activity. $A_{rs} = 19$	
	$N_{aer} = \frac{A_{rs}}{N_s} = 0.12$	
What is the amount of the removed entities at the PIM generation Phase ($E_{pim-domain}$)?	Number of Entities at the Domain Ontology. $Ne_{domain} = 64$	These data means that a large number of entities had to be removed from the domain. Although, none of the removed entities were out of the domain. They were all primitive properties missinterpreted as entities.
	$Ne_{pim} = 39$Number of Entities after the PIM generation.	
	$E_{pim-domain} = Ne_{pim} - Ne_{domain} = 25$	
What is the average number of inconsistent entities per number of removed sentences to fix such inconsistencies (N_{iprs})?	Number of inconsistent concepts at the domain ontology. $N_{ic} = 42$ (32 TBox, 10 ABox)	Our value indicates that we were able to keep more than half of the conflicting sentences, which means that once communication issues happened, we were able to keep some conflicting sentences. Not conflicting sentence had to be discarded.
	Number of removed sentences to solve the ontology's inconsistencies. $N_{rs} = 20$ (10 TBox, 10 ABox)	
	$N_{iprs} = \frac{N_{ic}}{N_{rs}} = 2.1$	

Table 4.10: GQM for Requirements Validation Issues

Requirements Validation Issues		
Question	**Metrics**	**Observations**
Which stakeholder was involved in the highest number of conflicts (S_c)?	Total amount Number of sentences. $A_s = 148$	Our process is able to track the stakeholder who has gave the largest amount of inconsistent statements. Such information is useful while dealing with validation issues helping to decide which stakeholder to interview if the provided requirements are inconsistent. Our processed showed that the Health_System_Specialist was involved at the highest number of conflicts. The 0.09 value showed that 9 percent of the sentences were inconsistent and were given by this specialist.
	$S_{Health\_System\_Specialist} = 13$(7 TBox, 6 ABox), $S_{Medical\_Specialist} = 8$ (2 TBox, 6 ABox), $S_{Security\_Specialist} = 1$ (TBox only).	
	$S_c = Max(\frac{S_{ci}}{A_s}) \simeq 0.09$.	

Which subject has the larger number of conflicts (Sub_c)?	Amount of sentences. $A_s = 148$	This number shows that the process is able to classify the subjects according to their level of inconsistency. Such data can be used to evaluate the complexity of relationships between entities at each subject. At our case study, the Health_System subject was the one with the highest index of inconsisntecies (9 percent of the amount of sentences.
	$S_{Health\_System} = 13$(7 TBox, 6 ABox), $S_{Medical} = 8$ (2 TBox, 6 ABox), $S_{Security} = 1$ (TBox only).	
	$Sub_c = Max(\frac{S_{ci}}{A_s}) \simeq 0.08$.	
What is the percentage of inconsistencies between facts about individuals and concepts(N_{iapis})?	The amount of the domain ontology concepts involved in ABox conflicts. $N_{icABox} = 17$	These numbers show that our process is able to distinguish between ABox and TBox inconsistencies. It also shows that the ABox inconsistencies were a small part of the total inconsistencies.
	The amount of conflicting individuals at domain ontology. $N_{ic} = 10$	
	The amount of conflicting sentences about facts whose individuals are involved at conflicts. $N_{cas} = 10$	

	The total amount of conflicting sentences. $A_{cs} = 34$	
	$N_{iapis} = (\frac{N_{cas}}{A_{cs}})*100\% \simeq 29\%$	
What is the percentage of inconsistencies between concepts(N_{itpis})?	The amount of concepts at the domain ontology involved in TBox conflicts. $N_{icTBox} =$	These numbers show that our process is able to distinguish between ABox and TBox inconsistencies. It also shows that the TBox inconsistencies were a great concern at the interview process.
	The amount of conflicting sentences about concepts that are involved at conflicts.$N_{cts} = 24$	
	$A_{cs} = 34$ The total amount of conflicting sentences.	
	$N_{itpis} = (\frac{N_{cts}}{A_{cs}})*100\% \simeq 70\%$	

Table 4.11: GQM for Traceability Issues

Traceability Issues		
Question	**Metrics**	**Observations**
What is the difference between the number of interviews conducted with the client's stakeholder and the	The amount of interviews given by the client's stakeholders. $Ni_{Cli} = 3$	This measure shows that no information about the interview process has gone missing during the first transformation

	The amount of interview tags at the domain ontology.$Ni_{Ont} = 3$	
	$Ni_{Cli-Ont} = Ni_{Cli} - Ni_{Ont} = 0$	
What is the difference between the number of interviews before, at the domain ontology and after the PIM generation ($Ni_{Ont-PIM}$)?	The amount of interviews given by the client's stakeholders represented as SDS tags at the domain ontology. $Ni_{Ont} = 3$	This measure indicates the same as the data presented before: we were able to keep track of all information from the interview process throughout the MDE generation process.
	The amount of interviews after the PIM generation. $Ni_{PIM} = 3$	
	$Ni_{Ont-PIM} = Ni_{Ont} - Ni_{PIM} = 0$	
What is the difference between the number of concepts inside each interview (entities marked with the same SDS tags) before, at the domain ontology and after the PIM generation ($Nci_{Ont-PIM}$)?	The amount of concepts defined at a j^{th} interview given by the client's stakeholders. $NciOnt_{Medical\_Specialist} = 22$, $NciOnt_{Health\_Specialist} = 33$, $NciOnt_{Security\_Specialist} = 9$	Our elevated value shows that a little bit of information related to the entities of the system may have gone missing. As a matter of fact, the missing information is related to the missinterpreted primitive properties, mapped to the PIM. If the PIM has not been "cleaned" this number tends to be 0

	The amount of concepts (classes) inside the j^{th} interview at the PIM built from the domain ontology. $NciPIM_{Medical\_Specialist} = 22$, $NciPIM_{Health\_Specialist} = 33$, $NciPIM_{Security\_Specialist} = 9$	
	$Nci_{Ont-PIM} = \sum_{j=1}^{n} NciOnt_j - NciPIM_j$, where n is the total amount of interviews.	

Following we summarize the relevant points of the case study analyzes:

In order to deal with scope issues, the most important data collected indicates that we were able to capture the same entities as the original version of the *Health Watcher*. The data also revealed the importance of the domain ontology cleaning activity. We have defined a higher number of entities at the domain, thus, our process may overestimate the scope of the designed software, so the activity of cleaning the domain ontology should be carried out more carefully as it can reduce the risk of overestimating.

Also to deal with scope issues, we showed that our process is able to provide some help while classifying the requirements complexity by showing the number of sentences related to each subject. Such data can be used to indicate the scope missdefinition risk for each requirement. Requirements containing less information have higher risk of suffering from scope issues.

In order to deal with communication issues, the most important data collected indicates that no information was lost during the MDE transformation process other than the information removed during the ontology cleaning activity. Again this

data draw our attention to the importance of this activity at our process. Also the process showed to be capable of removing inconsistent and contradicting sentences what definitely helps to provide a clear communication channel between teams.

In order to deal with validation issues, the most important data collected showed that our process is able to track the inconsistencies down and relate them to the stakeholder who has gave the statement that generated such inconsistency. Other than that we were fully able to make the separation of ABox and TBox inconsistencies and point different strategies to deal with each one.

In order to deal with traceability issues, the most important data collected proved that our process is able to keep information about the interviews at each level of the MDE process. It showed to be effective to help identify the stakeholders at the knowledge validation activity, providing the information needed to relate each inconsistency with the stakeholder who is responsible for it.

Chapter 5

Related Works

The related works were analyzed and compared to our proposal taking into account three criteria: methodology to represent requirements models, MDE compliance and employed technologies.

We have researched for a few works that use ontologies to represent the requirements model, such as [Knu04], where the authors propose the adoption of ontologies instead of UML semantics to model the domain of the system. Even though ontologies are powerfull enough to support the requirements representation and validation, as we have discussed before, such representation is hard for a regular requirements engineer to understand. Our multi-viewed approach deals with such need, defining three abstraction levels, among them one specific to represent requirements in a friendly fashion. Also, our proposal differs from [Knu04] for being MDE compliant, while their proposal only define a single level of abstraction, thus ignoring the fact that there are several profiles involved at a development process and that every single one of them has special needs.

Also, we have found a few works at the Natural Controlled Language field, such as [BLC+03, FdSBdPL03] where the authors establish a process for requirements gathering using NCL descriptions. Such approaches are, as far as we know, not compatible with MDE process since they are not model centered approaches. An MDE infrastructure lowers the coupling between the NCL and the requirements specification language. Also, the MDE infrastructure would help keeping the models synchronized once the transformations written are bidirectional

A few authors realized, like us, that one requirements engineering process is not able to supply the needs of all profiles at a development process by using a single view. We have studied a few works that also try to unify the use of ontologies and NCLs, like [DMO+06, BdPL04, SdPL06, dSdPLB04], where the authors propose a

process to build ontologies from natural language and apply them in the context of semantic web systems. All the authors adopt the *Language-Extended-Lexicon* (LEL) to represent the domain specifications. LEL divides text descriptions into notions and behavioral responses. The authors claim that ontologies are products built during a requirements engineering process. In our work, ontologies models represent domain knowledge and are used in several activities of the RE process. Also, LEL approach is less formal than our approach regarding the RE process, since we use a MDE approach to build ontologies, which imposes a better separation of responsibilities and modularization while these works still use simple XML parsing and processing techniques.

Requirements validation is also a concern and has been studied for a while. We have found a few works that, like ours, take advantage of the power of the first order logic beneath ontologies to help discovering inconsistencies. The work in [KS06] proposes the use of ontologies to represent the requirements knowledge in the context of requirements elicitation. This work proposes the use of automatic consistency checking tools to validate requirements models. The main difference between their work and ours is the use of MDE to convert the requirements into models to be used in further development phases, while theirs use ontologies only to check the requirements consistency and completeness. The authors also do not mention any specific methodology to build ontologies, what may turn the requirements analyst job harder.

Another alternative to NCL is a set of works that proposed the use of goal models to represent requirements. Goal models approaches are common and such models are very popular in RE. The work in [KL08] presents a comparative study with the most common approaches in goal models. The work classifies the contributions of goals-based approaches at each RE process activity. However, as far as we know, none of these works proposes the integration of goals-based approaches into a single RE process. Also, these approaches do not propose any mean of requirements validation. Furthermore, most of them do not use MDE nor natural language techniques. Another very important work in the area of goal modeling is the [vL08], where the author proposes a refining model to better model requirements through goals. The author also proposes the use of multiple views, linked through inter-model links constrained by rules for structural consistency. This is a very solid work but the author does not use any automatic technique to collect requirements. Also, the proposed derivation of the object model is not done by using any MDE technique. He also did not propose any mean to validate the requirements model.

We have also researched for works that use an MDE process to gather requirements and their usage as a Computation Independent Model. We have found a

very interesting work that unifies an MDE process for Requirements Engineering and NCLs [DMO+06]. The authors propose a process for requirements gathering through the usage of business models, also using NCL description to represent such models. Their work proposes a function to map descriptions in structured Portuguese into UML Use-Case diagrams. Our work differs from theirs in the chosen NCL. Our is much more general. Also, they were not worried in aligning their process with the ones specified at the RE literature. Also, the chosen Use-Case notation does not provide the formalism required to validate requirements models.

There are also another projects developed in the area of requirements specification that uses stronger logic formalism and MDE processes to represent requirement, like [GLGCGB05, GBLC05], that use a graph transformation approach integrated with an MDE process in order to build an architecture model from a feature model. These works differ from ours at the requirements specification notation and its expressiveness power. We have chosen to use NCLs for its simplicity and, still, power to represent and validate requirements. The choice for feature models restricts the expressiveness level, as such notation is used to model variability and common system features in family of systems. It does not allow the specification of relationship among system entities.

Finally, we have also examined works in the traceability area of knowledge. Among the researched articles, we would like to cite the [ARNRSG06] where the authors propose a complex and effective traceability scheme for model driven processes. Such work defines a general purpose metamodel to be used in traceability schemes. In that way, the work is more complete than ours. Nevertheless, the authors have not considered other requirements specification techniques/languages like the NCL or ontology. Their work is exclusively applied to UML model transformations.

Chapter 6

Final Remarks

At this section we will give a quick review of what we have presented throughout this work, discussing the obtained results and also indicating a few improvements points. We also have dedicated a single session to list the main difficulties encountered during the development of this work. By the end, we intend to present our plans for the future of this research.

6.1 General Comments and Case Study Evaluation

The research on the area of RE has grown fast in the last few years. In despite of this, there are still opened issues. In our work we researched the RE study area, looking for the main issues in RE process, we indicated those issues, listing the main initiatives that are trying to solve them. We also proposed ways to overcome these issues. To do so we presented a process and associated tool to improve current RE process through a MDE-based approach that represents requirements at several abstraction levels in order to communicate the knowledge among a greater number of involved user profiles, from both user and development teams view points.

The tool was built using state of art techniques and technologies. It makes possible to collect requirements knowledge direct from the stakeholders system descriptions. It also makes possible to transform this descriptions into lower abstraction level representations in order to meet the needs of the development team. Also, the presented tool builds representations that help the development team to manipulate and search within requirements in an easier way.

We have developed a case study to illustrate the execution of the proposed process and show that the developed RE tool is useful at the execution of the process. We believe we have acquired the desired goal and the process, with the help of the designed tool, was able to show hidden relationships among the entities and some inconsistencies.

In the case study we were able to transform a high level system description into a domain ontology, validate it, discover new relationships among their entities and finally derive, automatically, a PIM level UML class diagram using the XMI standard format.

In order to evaluate our case study we established a GQM [BCR94] plan, looking for improvement points and trying to quantify data about the execution of our process.

The results draw from our case study indicates, that a clear communication channel between teams minimizes most of the scope and communication issues, since both parts can express their view about the system's scope using a suitable notation. Moreover, the proposed process and tool provide a simple, but useful, traceability scheme, based on the idea of interviews, the most common requirements gathering technique. Our interview concept is very close to the idea of scenarios and actors proposed at [LB03]'s traceability model.

Our case study have also shown that our employed knowledge validation technique can be applied to help dealing with volatility issues. Once we were able to track down requirements inconsistencies by using ontologies and reasoning mechanisms. Thus, we provide some kind of insurance, in case the requirements change over the time (and they will), if the new requirement conflicts with some older one, the tool is able to alert the user. Our case study also showed that our approach can be used to integrate several views to represent RE knowledge, once no information was lost during the transformation.

6.2 Difficulties

The most difficult part of the development of this work was definitely to deal with the Model Driven Development tools. Since such area is constant development we encountered a large number of differences between different versions, like the definition of the UML 2.0 metamodel that presents a lot of changes from version 1.x to 2.x, mainly regarding the UML extensibility mechanisms (profile definition).

Also the ATL language is not an easy to learn language. The programmer have to be used to deal with functional languages in order to understand its semantics. Also the OCL library available at the ATL language is poorly documented. The used XML Injector did not make our job easy while writing the OWL parser. We came to the conclusion that this language was not designed to process text files, that is why we needed to pre-process our owl file before feeding it into the transformation engine.

Otherwise, the ATL syntax is not complicated at all and the iteration over collections scheme is very similar to the SmallTalk iterator scheme, which has proven to be an easy to use and highly productive scheme.

The study of the requirement engineering processes and opened issues also took a long time because requirements engineering is still an evolving area and a lot of works are being produced to deal with each one of the RE opened issues.

6.3 Future Works

In future works we plan to conduct a more complete study on ACE language's limits in order to better evaluate its capability to express requirements. We also intend to study ways to effectively specify the system dynamics, in order to provide a more complete requirements modeling process.

We have already started to search for suiting technologies and our primary studies points that the usage of a Business Rules approach may solve the problems to specify requirements, providing a notation fully understandable by stakeholders since a few implementations also use Natural languages.

At the development team level, works like [KLS08] have been studying the usage of BPMN to specify the business process of an enterprise and also providing transformation techniques in order to build a BPMN specification from a Business Rules Specification. We believe that such processes are close to the system dynamics requirements, so this notation should be studied in order to evaluate its simplicity to use and compatibility with MDE processes.

The GQM evaluation of our case study revealed the importance of the ontology cleaning activity. Such activity works as a filter, removing misinterpreted information from the concepts. This activity is definitely an improving point and at future works we will research for ways to represent primitive properties at NCLs. One way to deal with this is to extend the ACE notation, defining some specific namespace

for primitive properties. After that, the requirements engineers should be trained to identify primitive properties and use the specific namespace. We do recognize that such solution may degrade the reading level of the requirements specification.

We still found out that our approach using stereotypes for interview visualization may affect badly the PIM readability. At future works we will look for better ways to represent such information at the PIM level, perhaps using an UML 1.0 tagged value like approach.

Bibliography

[ALPT03] M. Alanen, J. Lilius, I. Porres, and D. Truscan. Realizing a Model Driven Engineering Process. Technical report, Technical Report 565, TUCS, Nov 2003, 2003.

[Alt00] R . Altwarg. Controlled languages: An introduction. *disponivel em:http://www.shlrc.mq.edu.au/masters/students/raltwarg/clindex.htm*, 2000.

[AMEL04] J.L.N. Audy, A. Majdenbaum, R. Espindola, and Lopes. Uma Análise Crítica dos Desafios para Engenharia de Requisitos em Manutenção de Software. *VII Workshop on Requirements Engineering*, 2004.

[ARE96] Amer Al-Rawas and Steve Easterbrook. Communication problems in requirements engineering: A field study, 1996.

[ARNRSG06] N. Aizenbud-Reshef, B.T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–526, 2006.

[AvH04] G. Antoniou and F. van Harmelen. Web Ontology Language: OWL. *Handbook on Ontologies*, 2:45–60, 2004.

[Bas05] V.R. Basili. Using Measurement to Build Core Competencies in Software. In *Seminar sponsored by Data and Analysis Center for Software*, 2005.

[BCR94] V.R. Basili, G. Caldiera, and H.D. Rombach. The Goal Question Metric Approach. *Encyclopedia of Software Engineering*, 1:528–532, 1994.

[BdPL03] K.K. Breitman and J.C.S. do Prado Leite. Ontology as a requirements engineering product. *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 309–319, Sept. 2003.

[BdPL04] K.K. Breitman and J.C.S. do Prado Leite. Lexicon Based Ontology Construction. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 19–34, 2004.

[BGMT06] JM Bhat, M. Gupta, SN Murthy, and I. Technologies. Overcoming Requirements Engineering Challenges: Lessons from Offshore Outsourcing. *Software, IEEE*, 23(5):38–44, 2006.

[BLC+03] B. Bryant, B. Lee, F. Cao, W. Zhao, C. Burt, J. Gray, R. Raje, A. Olson, and M. Auguston. From natural language requirements to executable models of software components. In *Proc. of the Monterey Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation*, pages 51–58, 2003.

[Boe84] Barry W. Boehm. Software engineering economics. *IEEE Trans. Software Eng.*, 10(1):4–21, 1984.

[Bro87] F.P. Brooks. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, 20(4):10–19, 1987.

[Bud03] F. Budinsky. *Eclipse Modeling Framework: A Developer's Guide*. Addison-Wesley, 2003.

[BvHH+04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, et al. OWL Web Ontology Language Reference. *W3C Recommendation*, 10:2006–01, 2004.

[BW84] VR BASILI and DM WEISS. A methodology for collecting valid software engineering data. *IEEE transactions on software engineering*, 10(6):728–738, 1984.

[C+99] J. Clark et al. XSL Transformations (XSLT) Version 1.0. *W3C Recommendation*, 16(11), 1999.

[CADW01] Ryan A. Carter, Annie I. Anton, Aldo Dagnino, and Laurie Williams. Evolving beyond requirements creep: A risk-based evolutionary prototyping model, 2001.

[CHCC03] J. Cleland-Huang, C.K. Chang, and M. Christensen. Event-based traceability for managing evolutionary change. *Software Engineering, IEEE Transactions on*, 29(9):796–810, Sept. 2003.

[CJB99] B. Chandrasekaran, JR Josephson, and VR Benjamins. What are ontologies, and why do we need them? 1999.

[CK92] M.G. Christel and K.C. Kang. *Issues in Requirements Elicitation*. Carnegie Mellon University, Software Engineering Institute, 1992.

[CRH+06] RM Colomb, K. Raymond, L. Hart, P. Emery, C. Welty, GT Xie, and E. Kendall. Version 3.3: The Object Management Group Ontology Definition Metamodel. *Ontologies for Software Engineering and Software Technology*, pages 1–25, 2006.

[DHR89] E. Dubois, J. Hagelstein, and A. Rifaut. Formal Requirements Engineering with ERAE. *Philips Journal of Research*, 43(4), 1989.

[DKW99] J.C. Derniame, B.A. Kaba, and D. Wastell. *Software Process: Principles, Methodology, and Technology*. Springer, 1999.

[DLM+08] N. Debnath, M.C. Leonardi, M.V. Mauco, G. Montejano, and D. Riesco. Improving model driven architecture with requirements models. *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on*, pages 21–26, April 2008.

[DLNS96] F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. *Principles of Knowledge Representation*, pages 191–236, 1996.

[DMO+06] F. Dias, G. Morgado, P. Oscar, D. Silveira, A.J. Alencar, P. Lima, and E. Schmitz. Uma Abordagem para a transformação automática do Modelo de Negócio em Modelo de Requisitos. *IX Workshop on Requirements Engineering (WERE'06)*, 2006.

[DPVOD02] J. Drew Procaccino, J.M. Verner, S.P. Overmyer, and M.E. Darter. Case study: factors for early prediction of software development success. *Information and Software Technology*, 44(1):53–62, 2002.

[dSdPLB04] L.F. da Silva, J.C.S. do P Leite, and K.K. Breitman. *C & L: uma ferramenta de apoio à engenharia de requisitos*. PUC, 2004.

[Egy03] A. Egyed. A scenario-driven approach to trace dependency analysis. *Software Engineering, IEEE Transactions on*, 29(2):116–132, Feb. 2003.

[ELPA05] R. Espindola, L. Lopes, R. Prikladnicki, and J.L.N. Audy. Uma Abordagem Baseada em Gestão do Conhecimento para Gerência de Requisitos em Desenvolvimento Distribuído de Software. *VIIIWorkshop on Requirements Engineering*, 2005.

[FdSBdPL03] C.H. Felicíssimo, L.F. da Silva, K.K. Breitman, and J.C.S. do Prado Leite. Geração de Ontologias subsidiada pela Engenharia de Requisitos. *sl. les. inf. puc-rio. br/cel/Artigos/Felicissimo_WER2003. pdf>*, 12, 2003.

[FHK+05] Norbert E. Fuchs, Stefan Höfler, Kaarel Kaljurand, Fabio Rinaldi, and Gerold Schneider. Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines. In Norbert Eisinger and Jan Małuszyński, editors, *Reasoning Web, First International Summer School 2005, Msida, Malta, July 25–29, 2005, Revised Lectures*, number 3564 in Lecture Notes in Computer Science. Springer, 2005.

[FKK08a] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for Knowledge Representation. In Cristina Baroglio, Piero A. Bonatti, Jan Małuszyński, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, *Reasoning Web, Fourth International Summer School 2008*, number 5224 in Lecture Notes in Computer Science, pages 104–124. Springer, 2008.

[FKK08b] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Discourse Representation Structures for ACE 6.0. Technical Report ifi-2008.02, Department of Informatics, University of Zurich, Zurich, Switzerland, 2008.

[Fra03] D.S. Frankel. *Model driven architecture*. Wiley New York, 2003.

[FSS99] Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto controlled english - not just another logic specification language. In Pierre Flener, editor, *Logic-Based Program Synthesis and Transformation*, number 1559 in Lecture Notes in Computer Science, Manchester, UK, June 1999. Eighth International Workshop LOPSTR'98, Springer.

[FST99] N.E. Fuchs, U. Schwertel, and S. Torge. Controlled natural language can replace first-order logic. *Automated Software Engineering, 1999. 14th IEEE International Conference on.*, pages 295–298, Oct 1999.

[GBLC05] B. González-Baixauli, M.A. Laguna, and Y. Crespo. Product Lines, Features, and MDD. *EWMT 2005 Workshop*, 2005.

[GDD05] D. Gašević, D. Djurić, and V. Devedžić. Bridging MDA and OWL Ontologies. *Journal of Web Engineering*, 4(2):118–143, 2005.

[GDDD04] D. Gašević, D. Djurić, V. Devedžić, and V. Damjanovic. Approaching OWL and MDA Through Technological Spaces. *3rd Workshop in Software Model Engineering (WiSME 2004), Lisbon, Portugal*, 2004.

[GDDS06] D. Gašević, D. Djurić, V. Devedžić, and B. Selic. *Model Driven Architecture and Ontology Development*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2006.

[GF94] O.C.Z. Gotel and C.W. Finkelstein. An analysis of the requirements traceability problem. *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94–101, Apr 1994.

[GF97] O. Gotel and A. Finkelstein. Extended requirements traceability: results of an industrial case study. *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, pages 169–178, Jan 1997.

[GGKH03] T. Gardner, C. Griffin, J. Koehler, and R. Hauser. A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the final Standard. In *MetaModelling for MDA Workshop*, 2003.

[GL93] J. A. Goguen and C. Linde. Techniques for requirements elimination. In *Proc. Int. Symp. Req. Engineering*, pages 152–164, Los Alamitos, California, 1993. IEEE CS Press.

[GLGCGB05] F.J.P. García, M.A. Laguna, Y.C. González-Carvajal, and B. González-Baixauli. Requirements variability support through MDD and graph transformation. *International Workshop on Graph and Model Transformation (GraMoT05), Electronic Notes in Theoretical Computer Science*, pages 161–173, 2005.

[Gog93] J. Goguen. Social issues in requirements engineering, 1993.

[Gor04] M. Gordon. Knowledge representation: Logical, philosophical, and computational foundations. *Distributed Systems Online, IEEE*, 5(1):9.1–9.3, 2004.

[GPFLC04] A. Gómez-Pérez, M. Fernández-López, and O. Corcho. *Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web*. Springer, 2004.

[HPSB+04] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member Submission*, 21, 2004.

[JAB+06] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. Atl: a qvt-like transformation language. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 719–720, New York, NY, USA, 2006. ACM.

[JFH92] W.L. Johnson, M.S. Feather, and D.R. Harris. Representation and presentation of requirements knowledge. *IEEE Transactions on Software Engineering*, 18(10):853–869, 1992.

[JK06] F. Jouault and I. Kurtev. Transforming Models with ATL. *LECTURE NOTES IN COMPUTER SCIENCE*, 3844:128, 2006.

[Kal08] Kaarel Kaljurand. ACE View — an ontology and rule editor based on Attempto Controlled English. In *5th OWL Experiences and Directions Workshop (OWLED 2008)*, Karlsruhe, Germany, 26–27 October 2008. 12 pages.

[KAMN98] C. Kamprath, E. Adolphson, T. Mitamura, and E. Nyberg. Controlled Language for Multilingual Document Production: Experience with Caterpillar Technical English. In *Proceedings of the Second International Workshop on Controlled Language Applications*, volume 146, 1998.

[KBW03] A.G. Kleppe, W. Bast, and J.B. Warmer. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional, 2003.

[Ken02] S. Kent. Model Driven Engineering. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 286–298, 2002.

[KF06] Kaarel Kaljurand and Norbert E. Fuchs. Bidirectional mapping between OWL DL and Attempto Controlled English. In *Fourth Workshop on Principles and Practice of Semantic Web Reasoning*, Budva, Montenegro, 2006.

[KL08] E. Kavakli and P. Loucopoulos. Goal Driven Requirements Engineering: Evaluation of Current Methods. In *Proc. 8th CAiSE/IFIP8*, volume 1, 2008.

[KLS08] S. Kherraf, É. Lefebvre, and W. Suryn. Transformation From CIM to PIM Using Patterns and Archetypes. *Proceedings of the 19th Australian Conference on Software Engineering - ASWEC*, pages 338–346, 2008.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the Association for Computing Machinery*, 42(4):741–843, 1995.

[KN05] R.S. Kaplan and D.P. Norton. The Balanced Scorecard: Measures that Drive Performance. *HARVARD BUSINESS REVIEW*, 83(7):172, 2005.

[Knu04] H. Knublauch. Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protege/OWL. *Proceedings of MDSW2004, Monterey, USA*, 2004.

[KS06] H. Kaiya and M. Saeki. Using domain ontology as domain knowledge for requirements elicitation. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 186–195. IEEE Computer Society Washington, DC, USA, 2006.

[LB03] J. Leite and K. Breitman. Experiences Using Scenarios to Enhance Traceability. In *2nd International Workshop on Traceability in Emerging Forms of Software Engineering at the 18th IEEE Conference on Automated Sofware Engineering, Oct*, 2003.

[Len96] Maurizio Lenzerini. Tbox and abox reasoning in expressive description logics. In *In Proc. of KR-96*, pages 316–327. Morgan Kaufmann, 1996.

[LHLHW+00] A. Le Hors, P. Le Hegaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document Object Model (DOM) Level 2 Core Specification. *W3C Recommendation*, pages 1–107, 2000.

[LPCD06] L.N. Leal, P.F. Pires, M.L.M. Campos, and F.C. Delicato. Natural MDA: Controlled Natural Language for Action Specifications on Model Driven Development. *LECTURE NOTES IN COMPUTER SCIENCE*, 4275:551, 2006.

[LS+99] O. Lassila, R.R. Swick, et al. Resource Description Framework (RDF) Model and Syntax Specification. 1999.

[MAJP02] S. Martin, A. Aurum, R. Jeffery, and B. Paech. Requirements engineering process models in practice. In *Seventh Australian workshop on requirements engineering (AWRE02), Melbourne, Australia*, pages 141–155, 2002.

[MBJK90] J. MYLOPOULOS, A. BORGIDA, M. JARKE, and M. KOUBARAKIS. Telos: Representing Knowledge About Information Systems. 8(4):325–362, 1990.

[McB04] B. McBride. The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS. *Handbook on Ontologies*, pages 51–66, 2004.

[Mel04] S.J. Mellor. *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley Professional, 2004.

[MFKH90] L. Macaulay, C. Flower, M. Kirby, and A. Hutt. USTM: a new approach to requirements specification. *Interacting with Computers*, 2(1):92–118, 1990.

[MGG+07] M. Milanović, D. Gašević, A. Giurca, G. Wagner, and V. Devedžić. Model Transformations to Share Rules between SWRL and R2ML. 2007.

[MM+01] J. Miller, J. Mukerji, et al. Model Driven Architecture (MDA). *Object Management Group, Draft Specification ormsc/2001-07-01, July*, 9, 2001.

[MM+03] J. Miller, J. Mukerji, et al. MDA Guide Version 1.0. 1. *Object Management Group*, pages 03–06, 2003.

[ND01] N Noy and McGuiness D. Ontology Development 101 - A guide to creating your first ontology. 2001.

[NE00] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *ICSE - Future of SE Track*, pages 35–46, 2000.

[NM00] N.F. Noy and M.A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 450–455, 2000.

[NMH03] E. Nyberg, T. Mitamura, and W.O. Huijsen. Controlled language for authoring and translation. *Computers and Translation-A Translator's Guide. Amsterdam/Philadelphia*, pages 245–281, 2003.

[NZP04] N. Nurmuliani, D. Zowghi, and S. Powell. Analysis of requirements volatility during software development life cycle. *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pages 28–37, 2004.

[PhS+06] E. Prud hommeaux, A. Seaborne, et al. SPARQL Query Language for RDF. *W3C Working Draft*, 20, 2006.

[PKCS95] K. Pulford, A. Kuntzmann-Combelles, and S. Shirlaw. *A quantitative approach to software management: the AMI handbook*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[Pre02] R.S. Pressman. *Engenharia de software*. McGraw-Hill, 2002.

[Pre05] R.S. Pressman. *Software Engineering: A Practitioner's Approach*. Boston, 2005.

[PWCC95] M.C. Paulk, C.V. Weber, B. Curtis, and M.B. Chrissis. *The capability maturity model: guidelines for improving the software process*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[RJ01] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *Software Engineering, IEEE Transactions on*, 27(1):58–93, Jan 2001.

[RW91] HB Reubenstein and RC Waters. The Requirements Apprentice: automated assistance for requirements acquisition. *Software Engineering, IEEE Transactions on*, 17(3):226–240, 1991.

[Sch06] D.C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *COMPUTER*, pages 25–31, 2006.

[SdPL06] L.F. Silva and J.C.S. do Prado Leite. Generating Requirements Views: A Transformation-Driven Approach. *Proceedings of the 3rd Workshop on Software Evolution through Transformations*, 2006.

[SGP98] G. Steve, A. Gangemi, and D.M. Pisanelli. lntegrating Medical Terminologies with ONIONS Methodology. *Information Modelling and Knowledge Bases IX*, 1998.

[Sin96] R. Singh. International Standard ISO/IEC 12207 Software Life Cycle Processes. *Software Process Improvement and Practice*, 2(1):35–50, 1996.

[SLB02] S. Soares, E. Laureano, and P. Borba. Implementing distribution and persistence aspects with aspectJ. *ACM SIGPLAN Notices*, 37(11):174–190, 2002.

[SM01] G. Stumme and A. Maedche. FCA-Merge: Bottom-up merging of ontologies. *7th Intl. Conf. on Artificial Intelligence (IJCAI01)*, pages 225–230, 2001.

[Som01] I. Sommerville. Software Engineering. 6th. *Ed., Harlow, UK.: Addison-Wesley*, 2001.

[SP90] A.P. Sage and J.D. Palmer. *Software systems engineering*. John Wiley & Sons, New York, NY, USA, 1990.

[ST04] R. Schwitter and M. Tilbrook. Controlled Natural Language meets the Semantic Web. In *Proceedings of the Australasian Language Technology Workshop*, pages 55–62, 2004.

[TBF+03] D.F. TOGNERI, SR BRITO, RA FALBO, OL TAVARES, and CS MENEZES. Um ambiente para aprendizagem cooperativa de engenharia de requisitos orientado a projetos. *Proceedings of the International Conference on Engineering and Computer Education (ICECE2003). Anais Eletrônicos... São Paulo*, 2003.

[vL08] A. van Lamsweerde. Requirements engineering: from craft to discipline. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 238–249. ACM New York, NY, USA, 2008.

[Wal05] E. Walker. Goal-question-metric (gqm) approach. Technical report, USA Department of Defense Information Analysis Center - DACS, 2005.

[WEC93] D.B. Walz, J.J. Elam, and B. Curtis. Inside a software design team: knowledge acquisition, sharing, and integration. *Communications of the ACM*, 36(10):63–77, 1993.

[WH96] R.H. Wojcik and H. Holmback. Getting a Controlled Language Off the Ground at Boeing. In *Proceedings of the First International Workshop on Controlled Language Applications*, pages 22–31, 1996.

[WHH98] R. Wojcik, H. Holmback, and J. Hoard. Boeing Technical English: An Extension of AECMA SE beyond the Aircraft Maintenance Do-

main. In *Proceedings: Second International Workshop on Controlled Language Applications (CLAW 98), Pittsburgh, PA*, 1998.

[Whi04] S. White. Process Modeling Notations and Workflow Patterns. *Workflow Handbook*, pages 265–294, 2004.

[WJJ06] S. Wang, L. Jin, and C. Jin. Ontology Definition Metamodel based Consistency Checking of UML Models. In *Computer Supported Cooperative Work in Design, 10th International Conference on*, pages 1–5, 2006.

Appendix A

Interviews

This Appendix presents the content of the interviews conducted during the Case Study.

Listing A.1: Interview with Medical Specialist

```
1  --Health_unit
2  Every health_unit allocates at least 1 specialty.
3  Every specialty is allocated by a health_unit.
4  Everything that allocates an specialty is a health_unit.
5  Every health_unit_data has a phone_number.
6  Every health_unit_data has a complement.
7  Every health_unit_data has a zip_code.
8  Every health_unit_data has a street.
9  Every health_unit_data has a city.
10 Every health_unit locates at least 1 doctor .
11 Every health_unit has medical_procedures .
12
13 --Disease
14 Every disease has at least 1 symptom.
15 Every disease is treated by at least 1 specialty.
16 Every specialty treats at least 1 disease.
17 Every disease has a description.
18 Every disease is a symptom_cause.
19 Every disease has a treatment.
20 Every treatment has drugs.
21 Every drug has at least 1 side_effect.
22 Everything that sells a drug is a pharmacy.
23 Everything that is treated is a disease.
```

```
24
25 --Symptom
26 Every symptom has a description.
27 Every symptom has a symptom_cause.
28 Everything that has a symptom is a disease.
29
30 --Patient
31 Every patient has a name.
32 Every patient has a phone_number.
33 Every patient has a complement.
34 Every patient has a zip_code.
35 Every patient has a street.
36 Every patient has a city.
37 Every patient has at least 1 symptom.
38 Every patient is treated by at least 1 doctor.
39 No patient is a disease.
40
41 --Doctor.
42 Every doctor has a name.
43 Every doctor has a phone_number.
44 Every doctor has a complement.
45 Every doctor has a zip_code.
46 Every doctor has a street.
47 Every doctor has a city.
48 Every doctor treats at least 1 patient.
49 Every doctor attends to at least one specialty.
50 Every doctor is located by at least 1 health_unit .
51 Every medical_procedure is executed by a doctor .
52 Everything that executes medical_procedures is a doctor.
53 Every doctor prescribes medical_prescriptions.
54 Everything that prescribes medical_prescriptions is a doctor.
55 Every medical_prescription has at least one drug.
56 No doctor is a nurse.
57 Every doctor has at least 1 specialty.
58 --Nurse
59 Every nurse has a name.
60 Every nurse has a phone_number.
61 Every nurse has a complement.
62 Every nurse has a zip_code.
```

```
63 Every nurse has a street.
64 Every nurse has a city.
65 Every medical_procedure is executed by a nurse.
66 Every nurse executes at least 1 medical_procedure.
```

Listing A.2: Interview with Health System Specialist

```
1 --Complaint
2 --complaint_status
3 Every complaint has a complaint_status.
4 Every opened_complaint is a complaint_status.
5 Every closed_complaint is a complaint_status.
6 Every suspended_complaint is a complaint_status.
7 Every complaint_status has a reason.
8 --complaint_status
9
10 Every complaint has a complaint_data.
11 --Complaint_data.
12 Every complaint_data has a complainer_name.
13 Every complaint_data has an observation.
14 Every complaint_data has a feedback.
15 Every complaint_data has a date.
16 --Complaint_data
17 Every responsible_employee is an employee.
18 Every complaint has a responsible_employee.
19
20 Every special_complaint is a complaint.
21 Every food_complaint is a complaint.
22 Every animal_complaint is a complaint.
23 --Complaint
24
25 --Food_Complaint
26 No food_complaint is a special_complaint.
27 No food_complaint is an animal_complaint.
28 Every food_complaint has an amount_of_affected_people.
29 Every food_complaint has a victim.
30 Every food_complaint has a location_of_treatment.
31 Every location_of_treatment allocates a specialty.
32 Every food_complaint has a suspicious_meal.
33 Every suspicious_meal has a suspicious_food_establishment.
```

```
34 Every suspicious_food_establishment is a problem_location_data.
35 No suspicious_meal is a complaint.
36 --Food_Complaint
37
38 --Animal_Complaint
39 No animal_complaint is a special_complaint.
40 No animal_complaint is a food_complaint.
41 Every animal_complaint has an amount_of_animals.
42 Every animal_complaint has a type_of_the_animal.
43 Every animal_complaint has a date_of_observation.
44 Every animal_complaint has a problem_location_data.
45 Every animal_complaint has at least 1 victim.
46 --Problem_location_data
47 Every problem_location_data has a phone_number.
48 Every problem_location_data has a complement.
49 Every problem_location_data has a zip_code.
50 Every problem_location_data has a street.
51 Every problem_location_data has a city.
52 --Problem_location_data
53 Every animal_complaint has an observer.
54 --Observer
55 Every observer has a name.
56 Every observer has a phone_number.
57 Every observer has a zip_code.
58 Every observer has a street.
59 Every observer has a city.
60 --Observer
61 --Animal_Complaint
62
63 --Special_Complaint
64 No special_complaint is an animal_complaint.
65 No special_complaint is a food_complaint.
66 Every special_complaint has a complement.
67 Every special_complaint has a city.
68 Every special_complaint has a district
69 Every special_complaint has a phone_number.
70 Every special_complaint has a street.
71 Every special_complaint has a zip_code.
72 Every special_complaint has an age.
```

```
73 Every special_complaint has an occupation.
74 Every special_complaint has a description.
75 --Special Complaint
76
77 --Citizen
78 If something X registers a complaint then X is a citizen.
79 Every complaint is registered by a citizen.
80 Every citizen has a name.
81 Every citizen has a street.
82 Every citizen has a zip_code.
83 Every citizen has a city.
84 Every citizen has a phone_number.
85 --Citizen
86
87 --Victim
88 Every victim has a name.
89 Every victim has a victim_data.
90 --Victim_data
91 Every victim_data has a phone_number.
92 Every victim_data has a zip_code.
93 Every victim_data has a city.
94 --Victim_data
95 --Victim
96
97 --Inferences
98 Everything that has a phone_number is a contact_data.
99 Everything that has a zip_code is a contact_address.
100 --Inferences
```

Listing A.3: Interview with Security Specialist

```
1 Every employee has a login.
2 Every employee has a password.
3 No victim has a password.
4 No victim has a login.
5 Every citizen has a login.
6 Every citizen has a password.
7 Every doctor has a login.
8 Every doctor has a password.
9
```

```
10 If X is a system_user then X has a login and X has a password.
11 If something X has a login and X has a password then X is a system_user.
12
13 Every system_user is a security_role.
14 Every query_security_role is a security_role.
15 Every update_security_role is a security_role.
16 Every no_access_role is a security_role.
17
18 Every citizen is a query_security_role.
19 Every employee is an update_security_role.
20 Every doctor is an update_security_role.
21 Every victim is a no_access_role.
22 Every observer is a no_access_role.
23
24 Every complaint is registered by an update_security_role.
25 Everything that registers a complaint is an update_security_role.
26
27 --Inferences
28 Everything that has a name is a person.
29 --Inferences
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo