FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Anwendung semantischer Technologien zur Unterstützung des Reportings in Projekten

Eine Fallstudie über Reports zur Konfliktanalyse und Kategorisierung von Anforderungen in einer realen Projektsituation

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Alexander Wagner
Matrikelnummer 0227425

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: ao. Univ. Prof. Stefan Biffl
Mitwirkung: Univ.-Ass. Dr. Matthias Heindl

Wien, 23.06.2010

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Technische Universität Wien

A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Using Semantic Technology to Support Project Reporting

A case study on requirements categorization and conflict analysis reports in a real-life project setting

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Alexander Wagner
Matrikelnummer 0227425

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: ao. Univ. Prof. Stefan Biffl
Mitwirkung: Univ.-Ass. Dr. Matthias Heindl

Wien, 23.06.2010

Erklärung zur Verfassung der Arbeit

Alexander Wagner
Guttmannstraße 14
A-2540 Bad Vöslau

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Bad Vöslau, 23.06.2010

Acknowledgments

I want to thank Professor Stefan Biffl and Matthias Heindl for guiding me through my master thesis.

I also thank Thomas Moser for his help in context with semantic technologies and Dietmar Winkler for sharing his experience regarding the conduction of evaluations.

Furthermore, I want to thank Siemens Austria for offering me the possibility to perform a case study in a professional software development setting. I want to thank especially Dr. Franz Reinisch for his extraordinary support and the possibilities he opened up for me.
I also want to thank my colleague, Michael Jaros, who supported me with his ideas and comments in accomplishing the goals of this master thesis.

Last but not least, I want to gratefully thank my parents for supporting me in my studies.

Abstract

Project Reporting is the process of acquiring specific information about the status of a project. Today's reporting solutions are able to fill a user-designed template with well-defined data from certain sources. A significant problem of traditional reporting systems is the lack of semantic meaning that's extractable from the data, like for example regarding its consistency.

Requirements engineering aims at keeping the set of requirements that must be fulfilled by a software product consistent and up to date throughout the project. Therefore requirements management tasks as requirements categorization and requirements conflict analysis are conducted. However, the manual conduction of these tasks takes significant effort and is error-prone.

My approach to realize a reporting solution that solves these problems makes use of ontologies as a semantic layer. Ontologies are formal, explicit specifications of concepts and their relations within a domain. By using ontologies, it is possible to infer new conclusions out of existing knowledge through automated reasoning.

I developed a semantic reporting approach and implemented a prototype called "OntRep".
And evaluated it in a real-world industrial setting at Siemens Austria to answer the following research questions:

- Which benefits does an ontology-based reporting approach bring regarding the categorization and conflict analysis of software requirements?
- What efforts have to be realized in order to prepare ontology-based reporting?

Major results were that OntRep provides reasonable capabilities for the automated categorization of requirements. It was considerably more effective to identify conflicts, and produced less false positives with similar effort compared to a manual approach.
The target audience of OntRep consists of everyone involved in software development projects who benefit from status information through reporting, but especially persons involved in requirements engineering and project management.

Abstract (German)

Reporting in Projekten bezeichnet den Prozess des Abrufs bestimmter Informationen über den Status eines Projekts. Die heutigen Reporting Anwendungen sind imstande, eine vom Benutzer gestaltete Vorlage mit exakt spezifizierten Daten von definierten Quellen zu füllen. Aber ein signifikantes Problem dieser traditionellen Reporting Systeme ist der Mangel an semantischer Bedeutung die aus diesen Daten gewonnen werden kann, wie zum Beispiel zur Überprüfung ihrer Konsistenz.

Requirements Engineering hat zum Ziel, die Anforderungen, die an ein Software Produkt gestellt werden, während des gesamten Projektverlaufs konsistent und auf dem neuesten Stand zu halten. Dazu führt das Anforderungsmanagements verschiedene Maßnahmen durch, wie die Kategorisierung und Konfliktanalyse der bestehenden Anforderungen. Die manuelle Durchführung dieser Aufgaben ist jedoch äußerst zeitaufwändig und fehleranfällig.

Mein Ansatz für eine Reporting Anwendung, die die genannten Probleme löst, macht sich Ontologien als semantische Informationsschicht zunutze. Ontologien sind formale, explizite Spezifikationen von Konzepten und deren Relationen innerhalb einer Domäne. Mit Ontologien ist es möglich, durch Reasoning neue Schlussfolgerungen aus bereits vorhandenem Wissen automatisiert zu ziehen.

Ich entwickelte ein Konzept für semantisches Reporting und implementierte dieses in einem Prototyp namens „OntRep".

Weiters evaluierte ich diesen in einer realen, industriellen Projektsituation bei Siemens Österreich um die folgenden Forschungsfragen zu beantworten:

- Welche Vorteile bringt ein ontologie-basierter Reporting Ansatz in Bezug auf die Kategorisierung und Konfliktanalyse von Softwareanforderungen?
- Welche Aufwände sind notwendig, um ontologie-basiertes Reporting vorzubereiten?

Die Ergebnisse zeigten, dass OntRep hervorragende Einsatzmöglichkeiten für die automatisierte Kategorisierung von Anforderungen bietet. Es war auch effektiver bei der Identifikation von Konflikten und erzeugte weniger falsch positive Resultate als der manuelle Ansatz.

Die Zielgruppe von OntRep besteht aus allen Personen in Softwareentwicklungsprojekten, die von Statusinformationen aus Reporting profitieren können, aber vor allem Personen involviert ins Anforderungs- und Projektmanagement.

Table of Contents

Figures

Tables

1. Introduction

In the context of this master thesis, knowledge can be defined as the awareness and understanding of all relevant facts, their implications and coherences within a certain situation. It is crucial for the quality of decisions in professional projects and companies.
In today’s business activities massive amounts of data are produced and gathered. Data, often saved in a machine-readable way, is the explicit representation of previously abstract information.
These loads of information could be used to gain a deep understanding about previous and ongoing activities, as well as to be able to forecast developments that will take place in the future.
In most cases the data itself is already existing. But it often has been created over several periods of time by various persons. Therefore it is spread over multiple locations with different technologies. This prevents the easy achievement of knowledge from it.

To address these problems and to provide a sophisticated way of gathering and analyzing data, the discipline of **Business Intelligence** has been elaborated.
According to Martin Grothe the notion Business Intelligence describes the analytical process which transforms fragmented data of a company into usable knowledge about the abilities, positions and aims of the observed fields [Grothe, 1999]. He also states that this process is being characterized by the benefits following from it and is no aim by itself, and the knowledge found can only become useful through purposeful application and communication.

Martin Grothe furthermore states that the Business Intelligence process can be divided into three phases (which are shown schematically in Figure 1) [Grothe, 1999]:

1. Delivery of quantitative or qualitative data in an either structured or unstructured way.
2. Identification of important relationships and patterns in the data.
3. Sharing and using the gained knowledge.

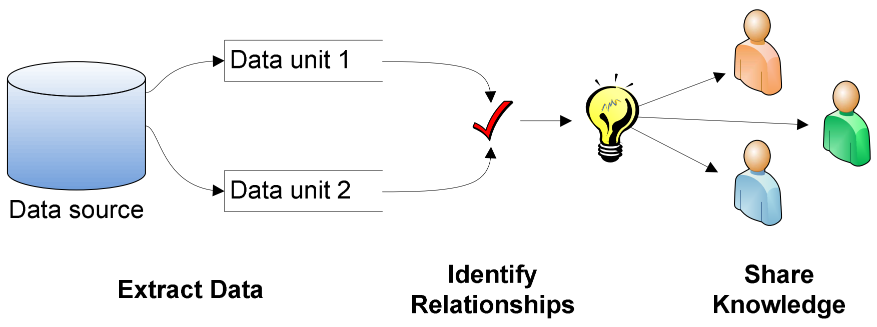


Figure 1: The Business Intelligence process, consisting of three phases

There are several techniques to perform these three steps of Business Intelligence. They are primarily performed in the enterprise and project context.
The focus for enterprises lies on the analysis of business performance and therefore mostly on financial, numerical data. Important techniques here are Data Warehouses, Data Marts, Multidimensional Models and Agents [Grothe, 1999].
For projects the focus lies on the monitoring of status and progress, which can be reflected in a wide variety of numerical and textual data. The traditional approach to flexibly extract data is Business Intelligence Reporting. Here the user can define reports, which are like templates that will be enriched with specified data when generating the report. This is the field of Business Intelligence the approach presented in this work focuses on.

Reports are documents that present extracted data from defined data sources in a structured way. They are usually not created manually, but by software systems. These systems were provided with the layout of the reporting document and access to the data sources the information comes from.
Usually, there are three tasks to perform to create a report document:

1. Define the report template.
2. Define the data sources to report from.
3. Generate the report document.

The task “Generate the report document” is typically performed fully automated by a software system, which itself has to perform three steps to complete it:

1. Extract the needed data from the data source.

2. Transform the data according to the report specifications.
3. Create a document using the report template and the transformed data.

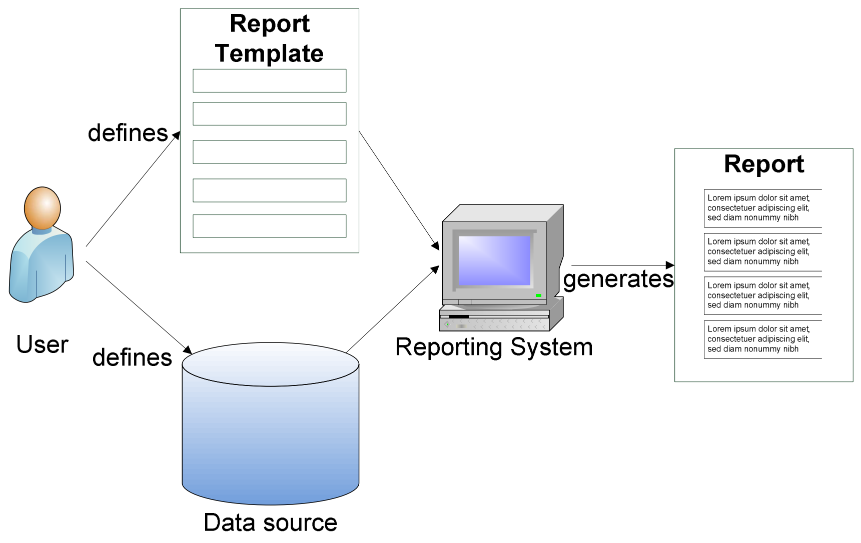

Report Template
defines
User
defines
Data source
Reporting System
generates
Report
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh

Figure 2: Using a reporting system to generate reports

Furthermore there are different types of data that can be used as a target for reporting systems [Chamoni and Gluchowski, 2006]:

- **Operative data**
  This form of information is the data that is being produced as a result of the business processes performed by all users. It usually is created by many different, heterogeneous applications and saved in multiple locations. Its primary aim is just the execution of the daily processes, most common during the processing of transactions and functions. They are often inconsistent and redundant, but no extra work is necessary to maintain these datasets. An example for operative data are the bug reports regularly created by the software developers.
- **Dispositive data**
  Dispositive data is produced and maintained especially for decision-supporting systems. It provides no use for the business processes, but is of high value for the management.
  Dispositive data is structured and related to specific fields and topics, there shall be no redundancies or inconsistencies. Furthermore it is essential to save this form of data over a period of time to be able to perform a historical analysis.

To use and maintain a dispositive database, it is necessary to take extra measures additionally to the usual business processes, and regularly extract (and possibly transform) data from the operative data sources.

An example for dispositive data is a separate database containing sales output numbers, which are regularly being calculated on the basis of different shop applications.

It is more difficult to create reports with high business value from operative data. When using dispositive data, lots of effort that increases the value of the information has already been done and can be reused in the reports.
Besides different types of reporting systems existing in the field of Business Intelligence, there are also distinct groups of users that have varying knowledge, expectations and tasks to fulfill: [ProClarity, 2005], [Kemper et. al., 2006]

On the top of a company, the *senior management* has to make decisions that have a large impact on the overall situation. Information for these decisions are of huge strategic value. They access a very high amount of data in a very aggregated form (meaning that it has been derived from lots of other sets of data). Normally they are not interested in the “raw” data, and want to create their own report layouts. Because they are technically unskilled in most of the cases, the reports have to be easily generated and reused with uncomplicated tools. Due to the high position of the management, they need to access the data from lots of different data sources at probably heterogeneous locations.

One level lower the *middle management* tries to keep awareness of smaller groups of organization. In the domain of software engineering, this would apply to project management, quality assurance or requirements engineers. They regularly need up-to-date facts and information of one or more project environments. These sets of data can partially be aggregated, but the middle management also needs more insight on the raw data itself than the top management. They also need customized reports, tailored to their specific needs, which can be generated regularly. It is very welcome for the project management if the reports they created for one environment can be reused in another project.

On the lowest level *users in directly productive activities* normally only need to access the raw data in its unaltered form. Because situations and tasks to make reports for vary a lot on

this level, the reuse of reports is very limited. It is more important here to be able to quickly and flexibly generate reports and adapt them to new tasks. Persons working on this business level are usually technically skilled and able to handle reporting systems that operate very near to the data source, without hiding technical complexities.
Though users on this level generally won’t need to be able to report from as many different environments as the management, it is quite possible that they need to access different, heterogeneous data sources, depending on the specific situation.
An example for a group of users on this level in the domain of software engineering would be the software developer, who might need to generate bug reports or retrieve build information. Another person on this level that benefits from Business Intelligence reporting is the software tester, who needs to analyze whole sets of current and previous test case results.

1.1 Problems of reporting systems

The use of Business Intelligence reporting methods can improve the processes of a company, whether at the domain of software engineering or other types of business. But there are some challenges and problems that users will encounter with traditional reporting solutions in certain situations:

- **Tight coupling of reports and data sources**

  Traditional reporting systems extract the data with which the report will be enriched directly from the data source. This data source and the access to it usually are specific for the project environment, and can be quite different in other projects. So when designing the report, one must define the exact way of accessing the data to fill the report with. This causes a tight coupling between the report and the data. When the user wants to reuse his reports in a different project with a technical different environment, many modifications have to be done first.

  An example could be a report that produces a structured list of all open tasks for the project. To do so, it accesses a database using several SQL queries. If the user wants to reuse this report now in a different environment, where the open tasks are not saved in the same way (due to the use of a different database management system, or even a completely different technique for persisting data), he first has to modify all queries in each report according to the target system, which can cause high efforts.

Besides the fact that these modifications are very cumbersome, depending on the tool he is using, the user could need to produce a copy of the report, meaning that he now has to deal with two instances of a report that actually fulfill the same purpose. This can cause inconsistencies and changes that have to be made in two locations instead of only one. Figure 3 shows an example situation:

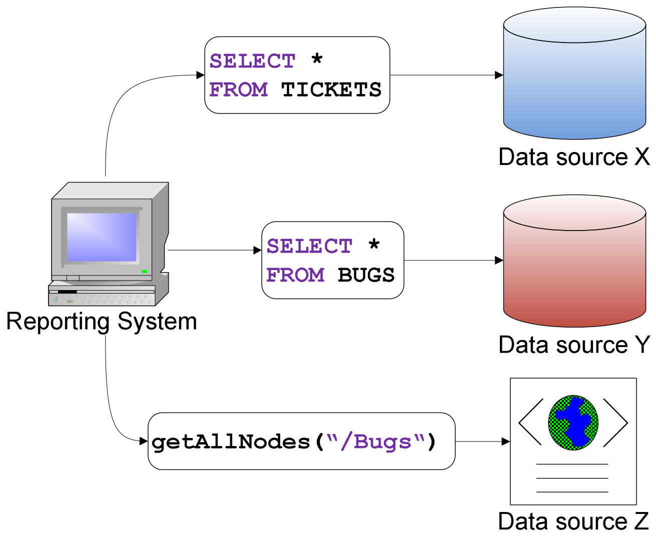


Figure 3: Technical access to different data sources can vary

- **Simultaneous use of reports for different environments**

Related to the previous issue that was stated here another problem arises: It can be very difficult to produce reports that simultaneously access heterogeneous data sources. In the worst case, the user has to generate a single report for each project environment, and then aggregate the data by hand. The user could also need to use several different reporting solutions, because of the different applications that have to be used for reporting in different environments. This would be a severe disadvantage for the top management, which usually is not as technically experienced as operative staff. But it is particularly the top management, which would especially be interested in overall statistics on a big scale, with highly aggregated data.

An example use case for this could be to find out about all hours of work done for several projects. With traditional reporting approaches it can be very time-consuming to first access all the timesheets (which could be saved in technically different ways) and then aggregate the numbers to get an overall overview. Furthermore it could be even more difficult to reuse the report that has originally been created for a single project in a multi-project setting.

The task to solve problems of this kind is known as *Data Integration*.

Figure 4 shows the idea of a reporting system that is able to aggregate data from different sources into a single report:

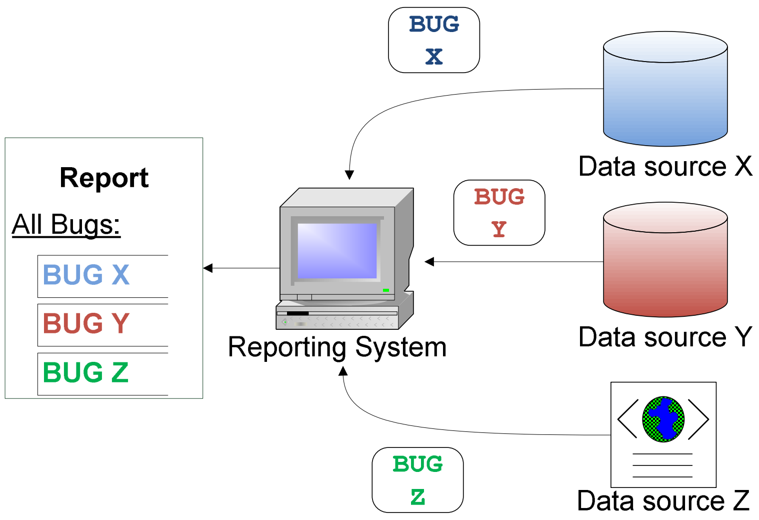


Figure 4: Simultaneously retrieving data from several sources for a single report

- **Lack of metadata**

The information structures that are in use in companies can become quite complex and difficult to understand. Data structures often evolve over several years through the usage of many different applications and other legacy systems.

This information is rarely self-descriptive. So even if the user knows what kind of information his report shall contain, it is another task to be able to find that information. He must know where to get the data he needs, interpret it in the right way and handle it according to its needs (for example through possibly necessary transformations).

A layer between the data and the reporting systems themselves, that includes information about the data records, could solve this problem. This metadata should describe the data and tell the user how to interpret and process it. It’s furthermore important that this layer is easily capable of being integrated into the reports. Such a layer could facilitate the work of an user that is not already familiar with the data storage tremendously. Especially for the management that does not automatically know which are the important sets of data to search for. Figure 5 shows a schematic example of an semantic “Info Layer”:

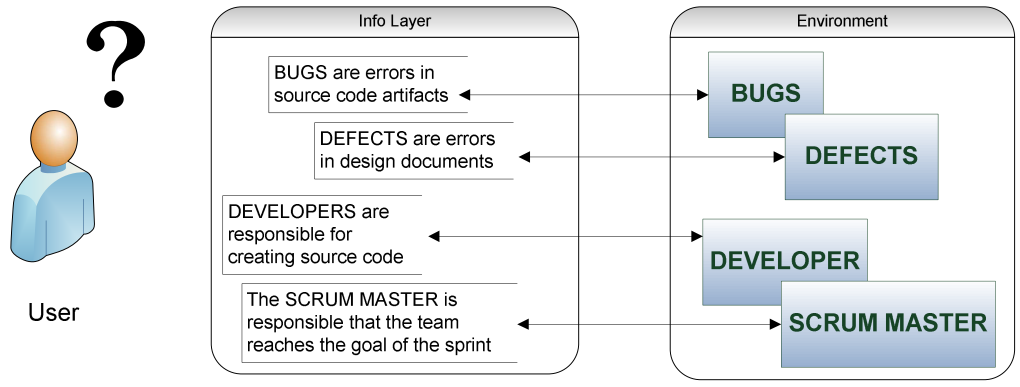

User
Info Layer
BUGS are errors in source code artifacts
DEFECTS are errors in design documents
DEVELOPERS are responsible for creating source code
The SCRUM MASTER is responsible that the team reaches the goal of the sprint
Environment
BUGS
DEFECTS
DEVELOPER
SCRUM MASTER

Figure 5: Via an information layer containing metadata, the user could be able to gain a better understanding of the environment

A layer containing metadata could not only be processed by a human user, but also by the reporting software itself. This way the software could gain semantic knowledge about the data and enrich the overall information value by using this knowledge for reporting purposes.

An example of how semantic information can be used for reporting will be presented in section 1.4 and is the main focus of the approach presented in this thesis.

Figure 6 shows how a reporting system accesses the semantic “Info Layer”:

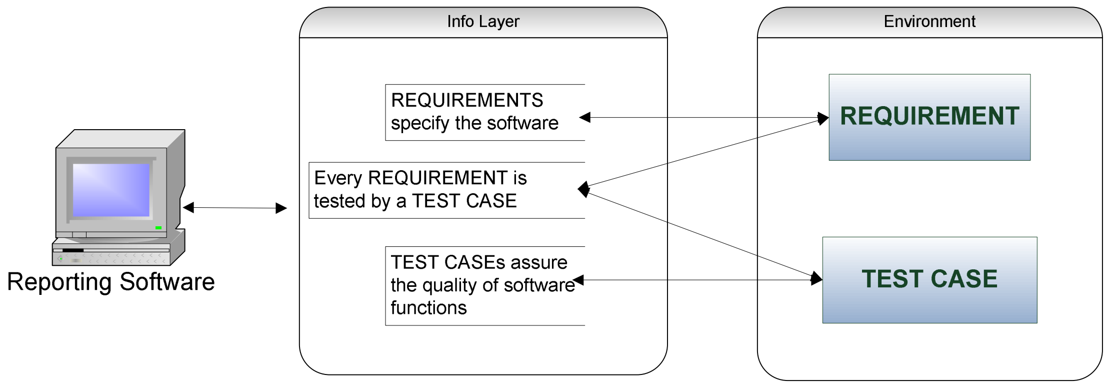

Reporting Software
Info Layer
REQUIREMENTS specify the software
Every REQUIREMENT is tested by a TEST CASE
TEST CASEs assure the quality of software functions
Environment
REQUIREMENT
TEST CASE

Figure 6: An information layer providing semantic information about the data to the reporting software

- **Difficult access to data sources**

  Related to the previous problem stated it is not only difficult to know where to get the desired information and how to interpret it in the right way. It is also the actual extraction step that can be a challenge. It is often necessary to define complex query statements (for example using SQL for databases or XPath for XML structures) to be able to get the necessary information. These queries can only be defined by technically

skilled users that are familiar with the underlying technologies. But this is no precondition that can be satisfied by the average user of a reporting system.
Generally there are two ways to solve this problem: Either provide a layer where the technical experts provide the technical details in an easily accessible way to all users. Or offer a special kind of interface that hides the complexities in the background and supports the user in the data extraction process. The latter is the main topic of a work by Spahn et. al. [Spahn et. al., 2008].

1.2 Benefits of using semantic technologies for project reporting

The combination of project reporting with semantic technologies can bring several benefits. As described in the previous section, an "Info Layer" could be used to semantically describe the concepts behind the data that is being reported from and enrich its overall information value. The semantic technology of *ontologies* provides the means to realize such a layer. Section 1.3 gives a more detailed introduction into ontologies.

The following figures compare a traditional reporting system with an ontology-based approach:

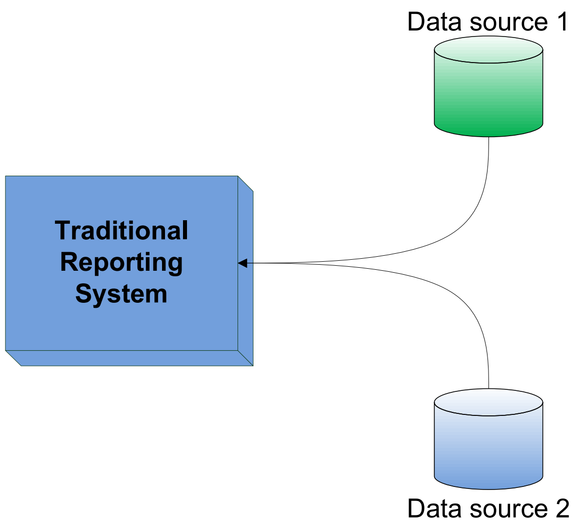


Figure 7: Traditional reporting system

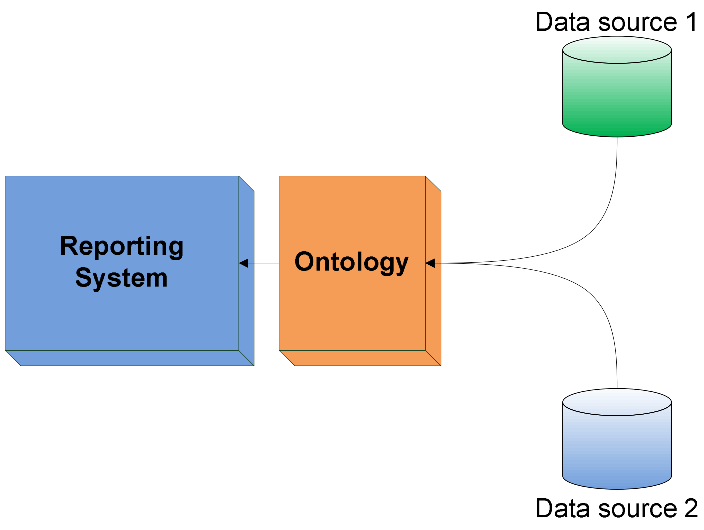


Figure 8: Ontology-based reporting

This traditional reporting system shown in the first figure directly accesses the sources it extracts data from. There is no instance that could provide the software with any additional knowledge about the data.

In contrast the ontology-based approach shown in the second figure: Here an ontology is introduced as a layer between the reporting software and the data. This ontology provides the software with information about how to interpret the data.

Now I want to discuss the benefits that this approach can bring:

1. **Usage of semantic knowledge in reporting software**

   The most obvious benefit of using ontologies is to enrich the reporting software with semantic knowledge about the data. Adding descriptive information about resources is one of the main purposes for using ontologies.

   When reporting does not only access the data itself, but also the metadata, the overall information value increases. But the software needs to be able to handle that metadata in a meaningful way.

   It is important to notice here that semantic reporting is not a single one application, but can be used very differently depending on the situation and the data. Ontologies provide the flexibility to describe a broad range of facts, so there are countless possibilities of how to make use of this knowledge.

   I will bring some practical examples for semantic reporting in section 1.4.

2. **Providing the user with information about the data**

   The knowledge that is being saved in an ontology can not only be processed automatically by reporting software, but also by the user himself.

   One cannot always assume that the user of the reporting system already knows the semantics behind the elements of the environment. This can be the case for project managers and other persons who are involved in the project.

   But people from “outside” the project (for example the top management, which is not an active member of the team, but can also need information about the project status) sometimes need to acquire information. It cannot be assumed that they know all the concepts of the project environment in detail.

An example situation: The software development team applies the agile process “Scrum” for its projects. Scrum introduces a set of terms and artifacts for the process, like for example “Product Owner”, “Scrum Master” or “Sprint Backlog”.

A person who is not familiar with this software development process does not know the exact meaning behind those concepts. He won’t know that he must search the “Backlogs” to get requirements for the product to develop, or at least has to spend efforts to find out.

An ontology describing the environment to report from could store human readable information about the concepts. The user could not only analyze the contents, but also the descriptions of the several concepts to find out what he needs.

3. Decoupling the reports from the data sources

The way to access different data sources can be quite different because of diverse technical implementations. Therefore it is possible that the user has to do lots of modifications when trying to migrate his existing report templates from one environment to another to be able to reuse them.

A simple example: The user has defined the following query to access all “Tickets” from his reports:

```
select *
from TICKETS
```

This query accesses a specific database table in a project environment. For migration of the reports to the second environment, where the tasks to work on are saved in the table “Issue”, the user has to change every query definition in every report template to the following definition:

```
select *
from ISSUE
```

He furthermore has to change each report to send the queries to another database than the one that was initially used.

This is a very simple example, in which I only assumed that the name of a table was different. In a more complex environment it can be the case that the whole structure of the two databases is different, causing more changes that have to be made. Like for example the joining of tables that becomes necessary.

The ontology-based approach introduces a layer that decouples the reports from the data sources, hiding the particularities of data access from the user. Figure 9 shows the approach in a schematic way:

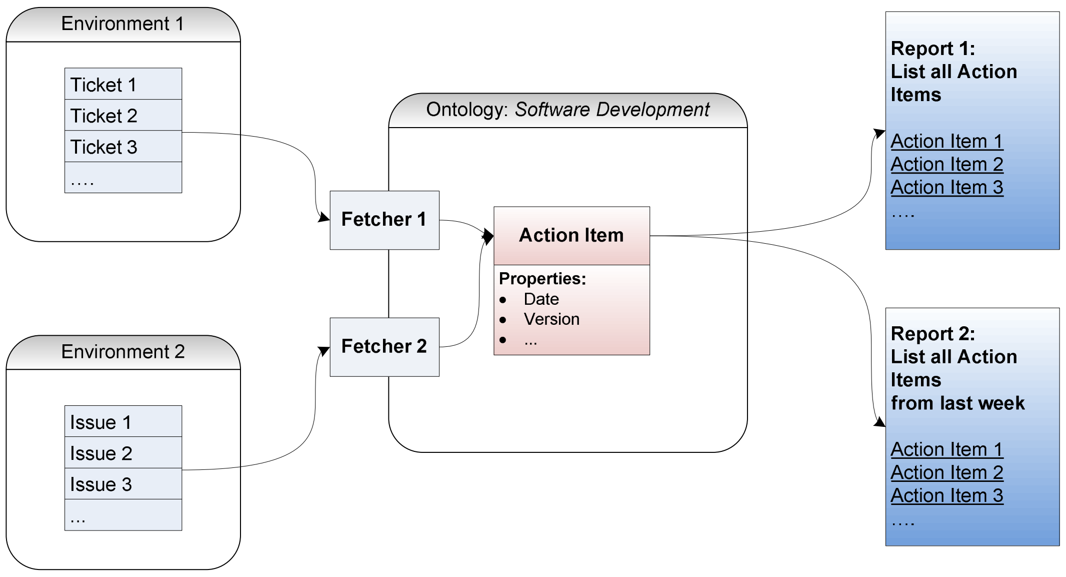


Figure 9: Indirect access to different environments using an ontology

As one can see in the figure the reports do not access the environments directly, but get the data from an ontology element named “Action Item”.
The ontology uses two specific “fetchers” to extract the data of the two environments. “Fetchers” are modules that fulfill the task of transferring the data from the data source into the ontology. The fetchers need to be configured in the example, so that “Action Item” maps to the database table “Tickets” in the first environment, and to the table “Issue” in the second environment.

4. Data integration: Combined reports for several environments

Another use case the user benefits from an ontology-based reporting system is that he is able to view the data from different environments in one single report. This way the user can obtain an integrated view on all data.
Figure 10 shows an example:

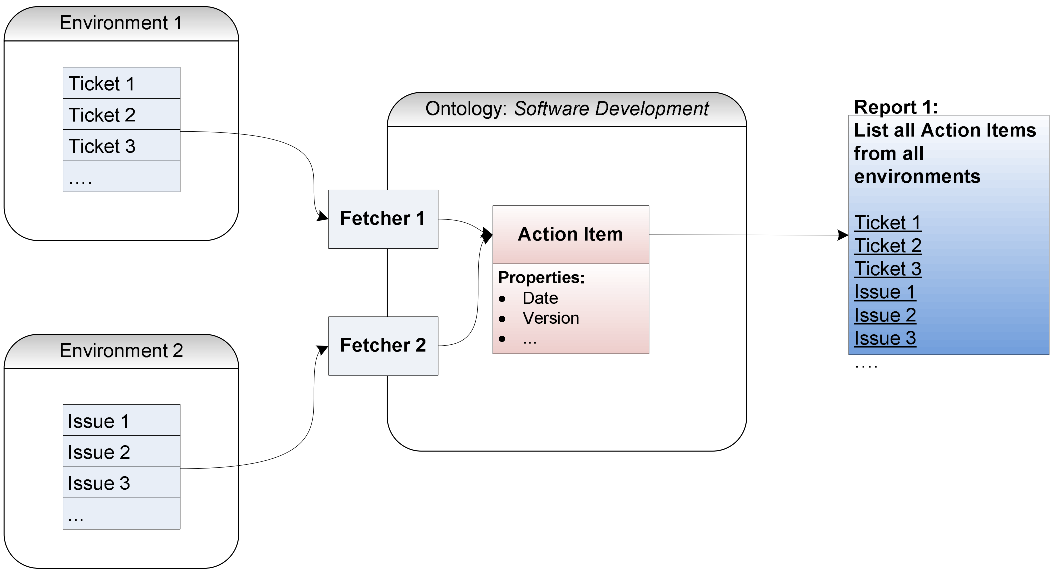


Figure 10: Ontologies can provide an integrated view on multiple environments

The diagram shows that the report can access the ontology class "Action Item", no matter from where its contents come from. The result is that the report lists all "Action Items" from all environments, including both "Tickets" and "Issues".

5. Providing reports for unskilled users

Another problem are difficulties when accessing the data sources. It can be necessary to construct complex query statements to be able to extract the desired data. This is especially a challenge for technically unskilled users.

An ontology model can be used to predefine complicated data access for other users. Each class in the ontology can be filled with data through the use of specific fetcher modules. So skilled users can define the technical details of data access by creating fetcher modules in advance.

Secondly, because an ontology model is very well suited to make a visual representation from it, it is an ideal candidate for a graphical user interface. This interface could not only provide ways to manipulate the ontology model itself, but also support the user to include the elements of the ontology into his report.

6. Provide a standard set of reports for a development process

By using ontologies as a descriptive layer between the reporting system and the data sources it can be possible to define a standard set of report templates that apply to a certain ontology, and use this predefined set in every environment the can be described by the ontology.

Applying this concept for software development projects: An ontology that maps the different concepts of a software development process could be used by every software company that applies this process. The ontology could be delivered with a ready-to-use set of report templates. These report templates would not have to be recreated from scratch for every new environment. They could be located at a centralized place, so that every team uses consistent reporting.

It would only be necessary to specify the fetchers to their data sources once. Each team could not only benefit from a system to generate self-defined reports, but also get a well-prepared set of reports for their process that are ready to be used. Figure 11 shows how two example report templates can be reused in every Scrum project:

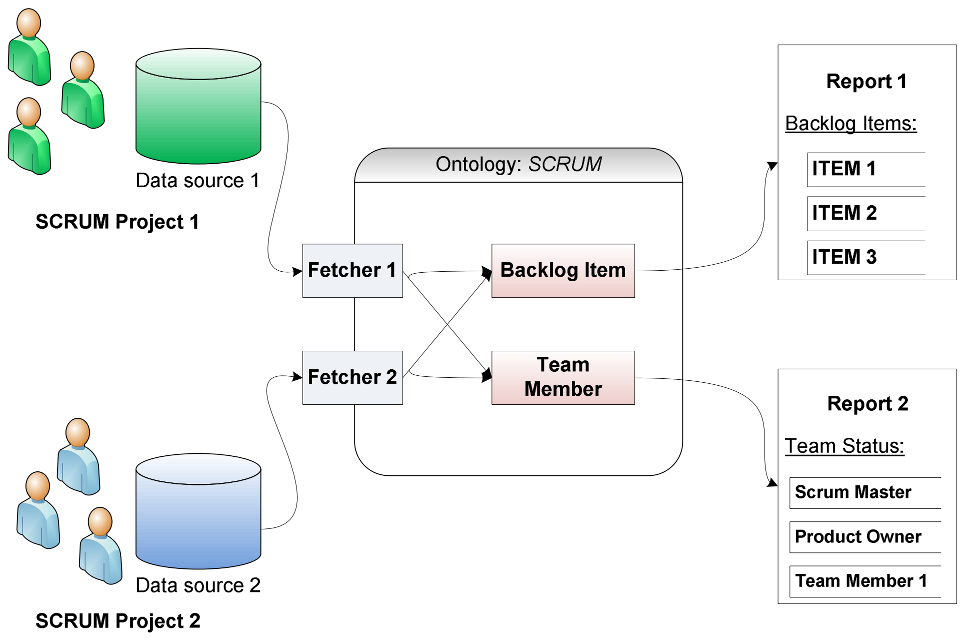


Figure 11: Providing ready-to-use report templates for software development processes

1.3 Introduction into ontologies

Originally the term "Ontology" describes a philosophical discipline which studies the nature of beings and reality, as well as their categories and relationships.

According to T. Gruber [Gruber, 1995] "*the term ontology refers to a formal description of the part of a reality*" in the field of computer science and knowledge management.

The most cited definition for “Ontology” in the sense of computer science has been declared by Gruber [Gruber, 1993]: “*An ontology is an explicit specification of a conceptualization.*” He furthermore states that “*For knowledge-based systems, what ‘exists’ is exactly that which can be represented.*”
After T. Gruber, an ontology contains a set of objects and describable relationships among them, reflected in a the vocabulary that can be used to represent knowledge [Gruber, 1993].

Pedranici et. al. bring the important aspect of reasoning and states that “*ontologies provide the means for describing the concepts of a domain of interest and the relationships between them in a way that amenable to automated reasoning*” [Pedrinaci et. al., 2008].

And Lula et. al. sum up the possibilities that ontologies provide [Lula et. al., 2008]:
“An ontology-based approach allows the analyst to:

- *represent the complex structure of objects,*
- *implement the knowledge about hierarchical structure of categories,*
- *show and use the information about relationships between categories and individual objects.”*

So ontologies provide the means to represent information about certain concepts and their relationships in a formal way that can be processed by machines.
There is no single ontology standard that exactly describes how ontologies are being defined. Many different formal languages for describing ontologies have evolved, fitting best for miscellaneous use cases.
Nevertheless the main components ontologies consist of are very common between different languages. In general they are:

- **Individuals (also referenced as Instances, Resources,...)**
  Individuals are the concrete elements that are being described in an ontology. Examples for individuals are the contact information for a particular person, a specific picture, book, or a software requirement. All these concepts could have a formal representation as an individual in an ontology.
- **Literals**
  Literals are data values of a specific type, for example of type “number” or “string”. Example literals are “Franz Joseph”, 42 or TRUE.

- **Relationships (also referenced as Properties)**

  Relationships describe possible connections between the elements of the ontology. The creator of an ontology can define relationship types, like for example an “*is-responsible-for*” relation that links a “*developer*” with an “*action item*”.

  There are some special kind of relationships that are already predefined in several ontology languages, like for example “*subClassOf*“.

- **Classes (also referenced as Concepts, Types,...)**

  Classes are abstract groups of objects, that share some kind of commonality. They represent the concepts that exist in the universe of the ontology and categorize groups of individuals.

  There is a special kind of relationship between two classes, that is usually constituted using a special notation, because it is very common: The “*is-a*” (or “*extends*”, “*subtype*”, “*generalisation*”, “*subClassOf*“, ...) relationship. This kind of relation states that one class is a “subclass” of another. So for example the class “*developer*” (containing all individuals that are developers) could be a subclass of “*team member*” (one could say: “*developer*” “*is-a*” “*team member*”).

1.3.1 Ontology description languages

Several standards have been developed in order to describe ontologies. I want to present the most common languages here that are of relevance for this master thesis [Jena, Hitzler, 2007]:

- **RDF**

  The “Resource Description Framework” [RDF] is a rather trivial language for describing ontologies. It is primarily intended for the definition of metadata about resources in the World Wide Web. In RDF, an ontology consists of a set of statements. All statements are composed of three parts: Subject, predicate, object. There are different ways to persist these statement-triples, one possibility is XML (called “RDF/XML”).

  RDF knows no hierarchic structure, its elements are not composed in trees, but in graphs. It has not been developed for hierarchical structuring of single documents, but to describe general relationships between different resources. Furthermore it is very easy to merge different graphs.

RDF also knows literals which can be used to describe the attributes of a resource. The following figure shows an example graph:

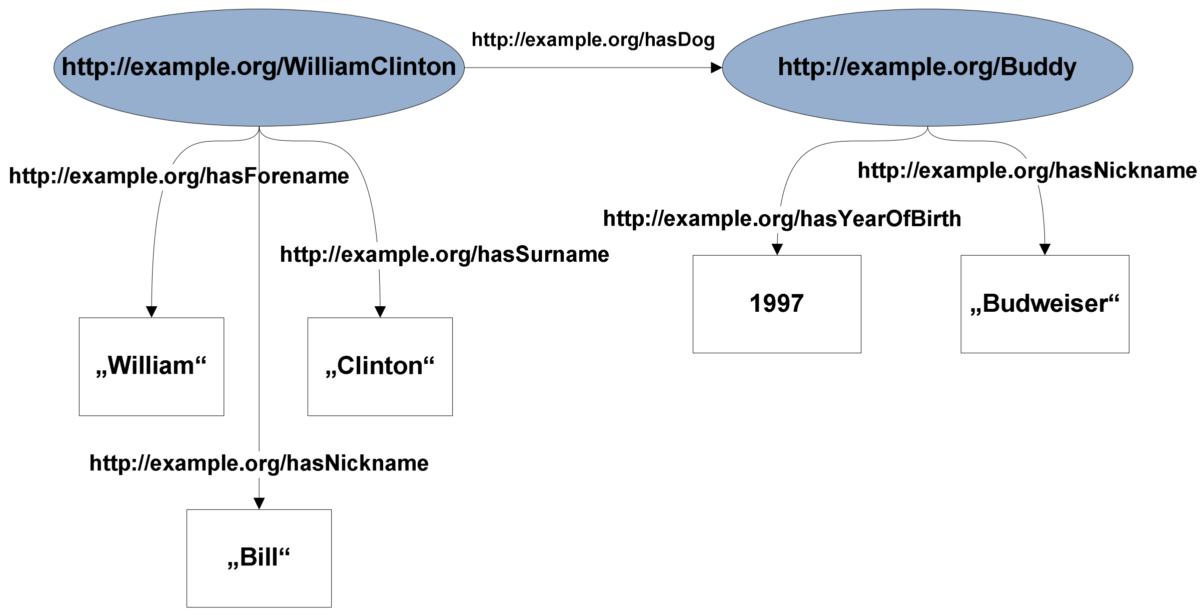

http://example.org/WilliamClinton
http://example.org/hasDog
http://example.org/Buddy
http://example.org/hasForename
http://example.org/hasSurname
http://example.org/hasNickname
http://example.org/hasYearOfBirth
http://example.org/hasNickname
1997
„Budweiser“
„William“
„Clinton“
„Bill“

Figure 12: RDF example graph

The example graph shows two resources that are being described, http://example.org/WilliamClinton and http://example.org/Buddy. It is important to know that these URIs do not necessarily have to point to valid resources on the internet. These URIs are only being used as identifiers for unique resources in the context of the ontology.
However it can make sense sometimes if the URI in an ontology points to an existing resource, especially when using RDF in combination with the web.

The example graph also shows that not only the resources are being identified through URIs, but also the relationships (the “edges” of the graph). This way a processing software knows that the literal “William” has different semantics than the literal “Clinton”.
Relationships can point to literals or other resources (like the relationship http://example.org/hasDog).

There are different ways to describe this graph in a formal way. One possibility is RDF/XML, which is often used in software solutions that process ontologies, but is hardly readable for humans.

Another alternative is to use the so-called “*Turtle*” syntax [Turtle]. In Turtle all statements are listed in the order subject, predicate, object. URIs can be abbreviated through the use of prefixes. The example graph has the following Turtle representation:

```
@prefix ex: <http:example.org/>

ex:WilliamClinton   ex:hasForename      “William” .
ex:WilliamClinton   ex:hasSurname       “Clinton” .
ex:WilliamClinton   ex:hasNickname      “Bill” .
ex:WilliamClinton   ex:hasDog           ex:Buddy .
ex:Buddy            ex:hasYearOfBirth   1997 .
ex:Buddy            ex:hasNickname      “Budweiser” .
```

- **RDF Schema**

RDF Schema [RDF-S] is an extension for RDF (implying that all valid RDF-S documents are also valid RDF documents).
With RDF a user can only describe knowledge about resources (or individuals). With RDF Schema it is also possible to define *terminological*, or *schematic* knowledge. So the user is able to define classes that group sets of individuals, and define the relationships between those classes (for example through the “subclass” relationship).
Furthermore it is possible to define hierarchies among relationships (also called properties). The http://example.org/hasDog relationship for example could be a subproperty to http://example.org/hasPet, implying that everybody who “has a dog” automatically “has a pet”.

When using RDF-S, the knowledge in an ontology can be separated into *terminological knowledge* (also called “TBox”), including all classes and their relationships, and the *assertional knowledge* (also called “ABox”), which consists of the knowledge about individuals.

We can now extend our example to make usage of RDF-S to describe the classes of the resources:

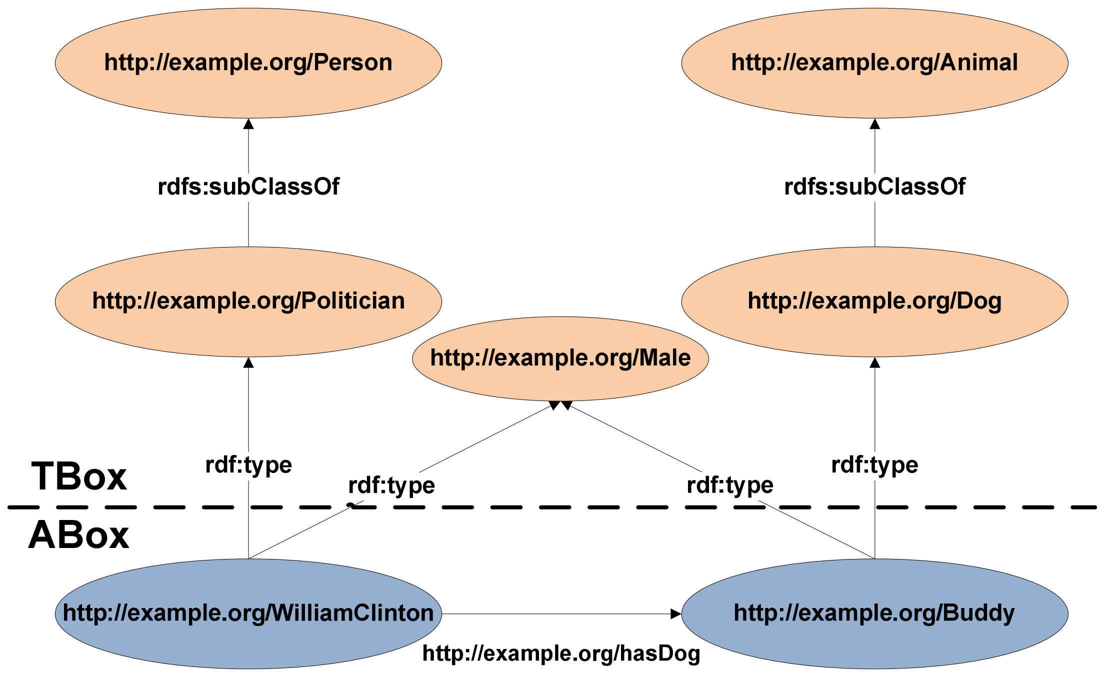


Figure 13: Using RDF-S to describe terminological knowledge

The resources of this example have now been related to classes. Classes can be ordered into hierarchies by using the (predefined) *rdfs:subClassOf* property, implying that every individual that is a member of the subclass automatically also is a member of the super class.

The example also shows that every resource can be a member of more than one class. Furthermore pay attention to the separation between terminological ("TBox") and assertional ("ABox") knowledge. This separation is not defined explicitly in an RDF-S ontology, but has been drawn into the example to illustrate the concept.

- **OWL**

The "Web Ontology Language" [OWL] (the different order of letters in the acronym "OWL" has been chosen on purpose) is based on first order logic and extends RDF-S. Everything that can be specified with RDF-S can also be described using OWL, which adds some predicates regarding properties and classes to the language.

OWL allows more complex ways to specify the concepts of an ontology. In RDF-S it is not possible to state that some fact is NOT the case in the context of the ontology. Due to the so-called *Open World Assumption* the absence of a statement does not imply that this statement does not hold true for the ontology.

In OWL there are possibilities to make such negated statements. Furthermore the user can make very specific statements about classes and their relationships:
For example class pairs can be marked as "disjoint", meaning that no single individual can be a member of both classes at once. In RDF-S it was not possible to define that no individual can be a member of the classes "smoker" and "non-smoker" simultaneously.
Two classes may also be stated as equivalent, meaning that they exactly contain the same individuals.
One can also define *enumerated classes* in OWL, which exactly contain the specified individuals, and none else.
OWL furthermore allows to make statements about the number of related elements of a class of individuals. For example one can state that "each project has at least one project manager", "each project has exactly 2 project managers", or "each project has at most 3 project managers".

OWL also allows to define special kinds of relationships: Properties can be declared as "symmetric", meaning that it automatically applies in both directions. For example if one individual is "the friend of" another, the "friend of" relationship holds true in both directions.
Properties can be declared as "transitive": Implying that if the property holds for x and y, and for y and z, it automatically holds for x and z . For example "located in" could be defined as a transitive property: Vienna is "located in" Austria, which is "located in" Europe, so Vienna is automatically also "located in" Europe.
Furthermore it is important to know that OWL has been divided into three sublanguages, which differ regarding their expressiveness: OWL Lite, OWL-DL and OWL Full, ordered by increasing expressiveness. Each sublanguage includes its less complex antecessor. The simpler subsets of OWL are intended to be easier processable to reasoning and description logic applications.

Besides the languages that are being described here, there are many other ontology languages that won't be described in more detail here. Examples are: DAML+OIL (the predecessor of OWL), KL-ONE (based on constructs called "frames"), KIF,... etc.

1.3.2 Ontology reasoning

One of the main benefits of using ontologies is the possibility to perform automated reasoning. Reasoning is the process of transforming implicit knowledge that is hidden in the knowledge base into explicit knowledge.
Explicit knowledge consists of all statements that have been explicitly defined by somebody using an ontology description language. They can be viewed instantly.
But implicit knowledge can only be found through analyzing all the explicit statements and conclude new statements out of the existing ones. This process is also called “inferring new statements”.

There are many rules in OWL of how to infer new statements. I will bring a few examples to provide a better understanding:

Transitivity is the origin of many reasoning operations. The “*subClassOf*” relationship for example is transitive:

- Dog is *subClassOf* Mammal
- Mammal is *subClassOf* Animal

→ Then we can infer: Dog is *subClassOf* Animal

The user is also able to define transitive properties by himself. If we define the “*isPartOf*” relationship as transitive and create the following statements:

- The wheel *isPartOf* the car
- The tire *isPartOf* the wheel

→ Then we can infer: The tire *isPartOf* the car

Important for our purposes are inference processes over OWL restrictions. These restrictions are logical constructs that can be used to restrict the individuals that are allowed to belong to a class. A sample restriction could be the statement “all members of the class ‘requirement’ must be tested by at least one ‘test case’”.
If there is enough information we can now infer through reasoning which individuals do or do not belong to a class with one or more restrictions. This process of classifying the individuals through reasoning is called “type-inference”.

Also see section 3.3.3, where practical examples are given of how to infer types by using restrictions.

There are also many other rules of how to infer new statements, that won't be covered in any more detail here.

1.4 Ontology-based reporting for Requirements Engineering

There are many potential benefits that arise from the use of ontologies for reporting purposes. This master thesis focuses on the usage of semantic knowledge for reporting software.

To be able to evaluate this concept, I chose to generate semantic reports on requirements engineering data, e.g. requirements categorization data, and requirements conflict data.
Requirements engineering is a discipline in software engineering that is concerned with the systematic elicitation and management of requirements that have to be met by a product in a software development project. So requirements engineering shall assure that the end product meets all customer's demands.
Without requirements engineering, it cannot be made sure that all stakeholders, including client, management and developers, share a common understanding of the product to develop. But this is a crucial condition to let all stakeholders be satisfied with the end product.

One of the main reasons why IT-projects fail nowadays are incompletely elicited or changing requirements. Requirements engineering shall help to handle these problems.

In the context of requirements engineering, there are three common tasks that can benefit from the usage of semantic reporting techniques:

- **Requirements categorization**
  The number of requirements for a big software project can become very high quite quickly. Therefore it is helpful to separate these requirements into meaningful groups.
  This way the project members that work on the product can keep an overview of requirements that are related. They can find common denominators between requirements and cross-cutting concerns (issues that are related to many requirements at once, like for example "Security").

It also helps in finding the points of intersection between different project teams, by checking the requirements that are related to more than one category. For example: If a requirement is in the categories “Security” and “Usability”, then it is a point of intersection between the team members responsible for security and usability. This can help to find out which topics shall be discussed together by different teams.

The groups or categories can be defined by the user. But it is not enough only to define these categories, the requirements have to be related somehow to the categories in a meaningful way.
Doing this process manually can become very time-consuming for a large number of requirements. Furthermore requirements often change which could create the need for an update of the categories and their related requirements.
A tool that automatically relates the given requirements into predefined categories by using natural language processing techniques could help reducing the necessary effort tremendously. Because of their open and easily extensible nature, ontologies suit well to serve as containers for requirements sorted into different categories.

- **Requirements conflict analysis**

Requirements that come from multiple different stakeholders can often include contradicting statements. This can be the case when different stakeholders do have conflicting requests that shall be fulfilled by the product, or when simply a mistake in the elicitation process has been made.
Furthermore there can be constraints in the context environment (e.g. technical constraints) that must be met by all requirements. It can also be the case that there are certain guidelines regarding the exact definition and documentation of requirements that must be fulfilled.

In all these cases it is important to check whether all requirements are free from conflicts with other requirements and all the constraints/guidelines at an early stage of the project. Conflicts found become more costly at later phases of a project.
Performing this check manually is a very time-consuming task, and scales up in an exponential way (every requirement has to be checked with all other requirements and constraints) with an increasing number of requirements.

Supporting this process with tool automation can help to reduce the necessary effort for the conflict analysis. Ontologies can serve as a technology to model the context environment and reasoning can help finding conflicts between different requirements or requirements and constraints.

- **Requirements tracing**

  Requirements tracing is the process of identifying dependencies and relationships between different requirements and between requirements and different kinds of artifacts. This way the lifecycle of a requirement can be traced from its origin in a specification document to its implementation in a source code artifact for example. There are many different kinds of traces, example relationship types are [MKS]:

  - Structure (“consists of”, “includes”)
  - History (“revision of”, “derived from”)
  - Conceptual traces (“fulfills”)
  - References (“defined through”, “composed in”)
  - Security (“authorized through”, “approved”)

  Requirements tracing can help to verify that a software product meets all goals that have been specified. Furthermore it is very helpful for the “change impact analysis”, where the consequences of the change of a requirement on all artifacts is being analyzed. This is important especially with unstable requirements that are likely to change often during the development process. It is also helpful for test coverage analysis, because traces can also link requirements with test cases.

  Nevertheless the efforts for manual establishment and maintenance of traces is considerably high, especially with an increasing number of requirements and artifacts. A solution that automatically identifies trace candidates could help to reduce efforts tremendously.

The remainder of this work is structured as follows: Section 2 outlines the related work in the fields of Business Intelligence, information integration and requirements categorization and conflict analysis.

Section 3 describes the OntRep approach in technical details. Section 4 deals with the methodology of the evaluation, while Section 5 presents its results. Section 6 discusses these results and Section 7 sums up this thesis and gives an outlook on possible future work.

2 Related Work

This section describes the related work on the fields of Business Intelligence, information integration and requirements categorization and conflict analysis.

2.1 Trends and ongoing work in the field of Business Intelligence

According to Gluchowski and Kemper the field of Business Intelligence is a growth market, as it can be seen in several reliable forecasts [Gluchowski, 2006]. They furthermore state that because of advancing globalization, world-wide spread of internet technologies and the increased information needs of stakeholders, the general conditions have changed massively. This causes discussions about how to extend or realign the field of Business Intelligence.

Gluchowski and Kemper spot three main trends [Gluchowski, 2006]:

- **Orientation towards business processes**
  There are ambitions to extend Business Intelligence with the direct analyzing of business processes. Data from processes is straightly being collected (often in real time), prepared and persisted. This way the relevant business processes can be investigated and improved. Furthermore it is possible to analyze the impacts of changes of the processes on the business and correlations between process tailoring and market success.
- **Enterprise Knowledge Management**
  Gluchowski and Kemper also state that the information earned from Business Intelligence analysis and also the models they are made of are not used sufficiently. They are often accessible only to a small group of users, though could be of interest for other persons in the company as well. Through descriptive meta information the data could be distributed and made usable for other stakeholders.
  They also mention that there are first approaches for the tight integration of Business Intelligence and Knowledge Management, but their implementations remain a big challenge.

- **Integrated development and conduction of Business Intelligence**
  Gluchowski and Kemper furthermore remark that traditional models of creation and maintenance of systems are not able to provide adequate solutions in the complex Business Intelligence environment. So people from science and industry are working on the creation and validation of new efficient models for development and conduction of Business Intelligence concepts. But they also note that much more effort has to be done to concretize and implement these models.

Mikroyannidis, Theodoulidis and Persidis set the main focus of their work on another issue that will undoubtedly become more important [Mikroyannidis et. al., 2006]: The discovery of Business Intelligence data from the internet.
In their paper they bring up the argument that the Business Intelligence solutions that are being built nowadays are very custom and specific for singular companies and cause high cost. Accessing the web could deliver newsworthy data that informs about industry developments. They present the "*PARMENIDES*" framework for information management on the web, which answers questions like:

- *"Is company X cheap? What is its position in the stock market?*
- *What is the collaboration profile of company X?*
- *What is the position of company X regarding its product pipeline?*
- *Who are (potential) clients of company X?*
- *How mature is the product line of company X?*
- *What is the quality of the management of company X?"*

Mohania and Bhide pay their attention on another topic that is becoming more important in the area of Business Intelligence: Information Integration [Mohania, Bhide, 2008]. They describe it as "*the category of middleware which lets applications access data as though there were in a single database.*"
It furthermore "*enables the integration of data and content sources so as to provide real-time read and write access, transform data for business analysis and data interchange, and data placement for performance, currency and availability.*"
This topic is especially important in companies, where data is being extracted from many heterogeneous data sources, using several different applications.
In their paper, Mohania and Bhide give an overview of the two basic approaches for Information Integration [Mohania, Bhide, 2008]:

- **Virtualization**

  In this approach a layer is being defined as schema, separating the user from each specific information source. The system is able to break down queries into sub-queries that are being sent to the appropriate data sources, combine their answers and return them to the user.

  Mohania and Bhide state that the main advantage of this approach is that the returned data is always current and up-to-date, while its primary challenge is to define the mapping between the virtualization layer and each data source.

- **Materialization**

  In this approach "*the data is materialized at the global level.*" This is done when using data warehouses, where there is no unstructured information. Mohania and Bhide say that "*the challenge in this approach is to determine the set of views which are to be materialized.*"

  And: "*Another problem in this approach is that of incremental view maintenance [...] When the underlying data sources are changing, there is a need for an efficient way to maintain the materialized view*."

Kohavi, Rothleder, and Simoudis identify the most important challenges in the field of Business Analytics in their article, and list currently emerging trends to face them [Kohavi et. al., 2002]:

- **Verticalization**

  To reduce the time needed for collecting, analyzing and reacting on enterprise data "*developers of analytical solutions started verticalizing their software, or customizing applications within specific industries.*"

  By "*incorporating industry-specific knowledge, companies are also able to optimize the performance of their applications for specific industries.*"

- **Comprehensible models and transformations**

  Kohavi et. al. state that "*Business users do not want to deal with advanced statistical concepts; they want straightforward visualizations and task-relevant outputs.*"

  They highlight the importance of solutions with understandable functionality and the benefits of visualizations.

- **Part of larger systems**

  The article also suggests that solutions shall be designed into whole systems. This issue typically affects the areas of data collection, generation and storage of unique identifiers (“*to help merge information and remove duplicate records*”), integration with multiple data sources and the need for hardware capable of handling large amounts of data.

- **New areas**

  Kohavi et. al. say that the success of business analytics for analyzing customer data has paved the way for new applications. They list three of them, that are particularly promising: Supply chain visibility, price optimization and work force analysis. With these kinds of applications companies are able to analyze their suppliers, their customers and also their employees.

- **Integration with action and measurement**

  Kohavi et. al. also state that two key questions become increasingly asked by business analytics users: “*How do I turn discovered information into action? and How can I determine the effect of each action on my organization’s business performance?*”

  They say that “*it is increasingly necessary that solutions use analytic results as a starting point toward the critical next steps of action and measurement*”.

  So it becomes more and more important that the usage of business analytics solutions directly results in measurements that increase business value.

2.2 Traditional reporting systems

There are lots of systems available to perform Business Intelligence reporting. Vega presents a Java based solution that has been built for usage in an university setting with a very heterogeneous environment [Vega, 2001]. They built their application upon the commercial reporting product “JReport”, which is based on Java and quite feature-abundant [Jinfonet].

Vega also says that *“we are living in the middle of the thin client computing. Servers are dealing again with the main part of the work.”* This is the reason why they relied on an approach where the main application logic is being processed by a “Tomcat” [Tomcat] web server using “Java Server Pages” [JSP] for dynamic page generation. This way everyone that is allowed to access the web server can create reports using a web browser.

This is a trend that is also identified by Keller [Keller, 2004]. He states that the appliance of internet technologies for reporting tools eases the connection of many users to the system.
Keller furthermore notes that the main application area for Business Intelligence tools is reporting and analysis, because it fulfills the main needs of the users for quick and easy access to indicator numbers.
He also separates the available tools into two classes [Keller, 2004]:

- **Production reporting (also called "static reporting" or "operative reporting")**
  Users of these tools regularly get reports sent without creating new queries. These "passive report receivers" make up 70% of the tool licenses.
- **Business Intelligence reporting**
  The main difference to production reporting systems is that they allow the user to manipulate the reports himself.

While Keller focuses on commercial products for Business Intelligence reporting, Kleijn sees the upcoming of open source solutions [Kleijn, 2006]. Besides products for Data Warehousing, ETL-processing ("Extraction, Transformation & Loading") and OLAP systems ("Online Analytical Processing"), the article names some applications for reporting in particular:
The software company "JasperSoft" created the reporting engine "JasperReports" [JasperReports]. They also offer a graphical designer which can be used for their reporting tool called "iReport".
"Pentaho" is another active company in the field of open source Business Intelligence [Pentaho]. They provide many different solutions and combine them to a whole Business Intelligence framework.

Another open source project that has drawn lots of attention in the community is "Eclipse BIRT". It was first proposed by the Actuate corporation [Actuate] in 2004 and became a top level project in the Eclipse community.

2.3 Ontology-based information integration and reporting

Several works exist that try to use ontologies as a mean for data integration (or "information integration"). According to Lenzerini [Lenzerini, 2002], "*data integration is the problem of* *combining data residing at different sources, and providing the user with a unified view of these data.*”

Wache et. al. provide a survey of existing approaches for ontology-based information integration. They sum up the most important uses of ontologies [Wache et. al., 2001]:

- **Content explication**

  The most common use of ontologies is to provide a formalized description for the semantics of the information sources.
- **Query model**

  The ontologies can be used as a global query schema. Users can be allowed to formulate queries by using the concepts of the ontology, which are being processed by the system into appropriate sub-queries that fit for each source.

  Wache et. al. see “*the advantage that the structure of the query model should be more intuitive for the user because it corresponds more to the user’s appreciation of the domain.*”

  But Wache et. al. also remark that “*from a database point of view this ontology only acts as a global query schema. If a user formulates a query, he has to know the structure and the contents of the ontology; he cannot formulate the query according to a schema he would prefer personally. Therefore, it is questionable where the global ontology is an appropriate query model.*”
- **Verification**

  Mappings from a global to the local information source schema have to be specified. Their correctness can be improved considerably if they can be verified automatically. According to Wache et. al., a “*sub-query is correct with respect to a global query if the local sub-query provides a part of the queried answers, i.e. the sub-queries must be contained in the global query (query containment) [Calvanese et al., 2001; Goasdoué et al., 1999].*”

  The ontologies contain all the information necessary to be able to validate the mappings with respect to them.

Wache et. al. furthermore identified three different directions in these approaches of how to realize ontology architectures [Wache et. al., 2001]:

- **Single Ontology approaches**

  This approach uses one global ontology with a shared vocabulary that specifies the semantics. All sources of information are linked to this global ontology.

  This method can be used "*where all information sources to be integrated provide nearly the same view on a domain. But if one information source has a different view on a domain, e.g. by providing another level of granularity, finding the minimal ontology commitment [Gruber, 1995] becomes a difficult task.*"

  Wache et. al. also point out another disadvantage of single ontologies: "*Also, single ontology approaches are susceptible to changes in the information sources which can affect the conceptualization of the domain represented in the ontology. Depending on the nature of the changes in one information source it can imply changes in the global ontology and in the mappings to the other information sources.*"

- **Multiple Ontologies**

  In approaches with multiple ontologies, each information source is defined by its own ontology. So the ontology that is being used can be a combination of several other ontologies, which are not obliged to share the same vocabulary.

  Though this fact makes this approach less susceptible to changes in information sources, Wache et. al. highlight that "*in reality the lack of a common vocabulary makes it extremely difficult to compare different source ontologies.*"

  To solve this problem, it is necessary to define inter-ontology mappings that specify semantically corresponding terms of different ontologies. This mapping also has to consider possibly different views on a domain, which could for example result in different granularities of concepts in multiple ontologies. Wache et. al. believe "*that in practice the inter-ontology mapping is very difficult to define, because of the many semantic heterogeneity problems which may occur.*"

- **Hybrid Approaches**

  To be able to cope with the problems of single or multiple ontology approaches, hybrid approaches have been developed. There the information of each source is described in its own ontology, just like with multiple ontology methods.

  But additionally, there is a shared vocabulary that contains the basic terms of a domain, and makes the source ontologies comparable to each other [Goh, 1997; Wache et al., 1999]. There are several possible ways how this can be accomplished [Wache et. al., 2001].

Wache et. al. sum up the pros and cons of this approach: *“The advantage of a hybrid approach is that new sources can easily be added without the need of modification in the mappings or in the shared vocabulary. It also supports the acquisition and evolution of ontologies. The use of a shared vocabulary makes the source ontologies comparable and avoids the disadvantages of multiple ontology approaches. The drawback of hybrid approaches however, existing ontologies cannot be reused easily, but have to be re-developed from scratch, because all source ontologies have to refer to the shared vocabulary.”*

Maurizio Lenzerini highlights the importance of the specification of the correspondence between the data at the local sources and those in the global schema when designing a data integration system [Lenzerini, 2002].
He furthermore describes the two basic approaches that have been suggested in literature to specify this mapping:

- **Local as view**
  This approach has the aim to define a view from each local source onto the global schema.
  It is the method that is used usually for systems that are based on ontologies or similar models. Lenzerini notes its advantages [Lenzerini, 2002]: “*This idea is effective whenever the data integration system is based on a global schema that is stable and well-established in the organization. Note that the LAV approach favors the extensibility of the system: adding a new source simply means enriching the mapping with a new assertion, without other changes.*”
  But query processing can pose quite a challenge in this approach, because “*the only knowledge we have about the data in the global schema is through the views representing the sources, and such views provide only partial information about the data. Since the mapping associates to each source a view over the global schema, it is not immediate to infer how to use the sources in order to answer queries expressed over the global schema.*” [Lenzerini, 2002]
- **Global as view**
  Contrary to “local as view”, which “starts” at the global schema, this method “starts” with the local sources. For each information entity, a view is being defined onto the appropriate data sources.

This method is especially effective for sources that are stable and don’t need to be changed. It is primarily used in approaches that use materialization (also see section 2.1), for example Data Warehouses.

Lenzerini notes the main advantage of this approach: “*Note that, in principle, the GAV approach favors the system in carrying out query processing, because it tells the system how to use the sources to retrieve data.*”

He furthermore remarks that extending the system with new sources is a big problem when using this approach: “*The new source may indeed have an impact on the definition of various elements of the global schema, whose associated views need to be redefined.*”

Saggion et. al. present another system that uses ontologies to be able to extract and gather Business Intelligence information [Saggion et. al., 2007]. They use domain ontologies that have been developed with the help of domain experts and implemented in OWL.

Saggion et. al. focus on textual information as data sources. So they apply natural language processing (NLP) techniques to populate their knowledge base. Their application is part of the EU Musing project with the goal to gather international company information.

Spahn et. al. present another interesting work in the area of ontology-based Business Intelligence [Spahn et. al., 2008]:

One of the main problems they try to address is that technically unskilled users have difficulties when trying to flexibly retrieve needed data in an ad-hoc way. They often have to rely on information that can be retrieved from queries and reports that have been predefined for them by IT experts. The ontology-based architecture and tool that is presented by Spahn et. al. shall enable users to easily access data and create queries themselves without any need for IT experts.

They build on a semantic middleware that can integrate data from heterogeneous data sources using a business level ontology to provide a single view onto the system. They furthermore developed the “Semantic Query Designer” that allows users to navigate conveniently through the information and flexibly build queries.

2.4 Related work on automated categorization of requirements

Automatically analyzing the contents of text and identifying certain characteristics is a huge topic in computer science in general. It is being researched in the field of “Natural Language Processing”. Techniques from natural language processing are very important to be able to relate requirements that are described in a human readable language with a set of given categories.

An important technique that is being used for natural language processing is “part-of-speech (POS) tagging”. This technique allows to identify the grammatical categories of different words in a sentence. These categories can be for example “noun”, “verb”, “adjective” or “adverb”. The tagging process is based on the definition of the words themselves as well as the context they are in. Additionally, parsers can transform the text into a machine-readable data structure (also called “parse tree”) that shows the grammatical structure and hierarchy of the input text [Choi, 2000].

Another useful tool in natural language processing are lexical databases. “WordNet” is a large lexical database of the English language [WordNet, 1995]. It does not only contain the English vocabulary, but also groups nouns, verbs, adjectives and adverbs that are related to the same concept into sets of cognitive synonyms, so-called “synsets”. The objects of these synsets are interlinked with each other by means of conceptual-semantic and lexical relations. More details on how WordNet is being used in our approach can be found in section 3.2.3.

A field of ongoing work related to requirements categorization is called “Aspect-oriented requirements engineering”. This field deals with the identification and analysis of crosscutting requirements, thus different requirements that share a relationship to a common topic. A typical example for such a crosscutting concern is “Security”, usually related to a whole set of different requirements.

Chitchyan et. al. present a tool-suite to support the whole aspect-oriented requirements engineering process [Chitchyan et. al., 2006; Sampaio et. al., 2005]. They first analyze the grammatical structure of the requirements text using natural language processing methods. Then a tool developed by them called “EA-Miner” is responsible for identifying crosscutting concerns. It analyzes the contents of requirements and tries to find common aspects that are semantically related. It also differentiates between non-functional concerns (by accessing a

lexicon of non-functional aspects in requirements engineering) and functional concerns (by checking for repeated occurrences of process words in different requirements). After some more refinement steps the results are being arranged and can be presented to the user.
They also did a number of case studies that showed the high scalability and efficiency of the tool suite. According to Chitchyan et. al., the results produced by it are comparable to a requirements engineer performing the same tasks.

2.5 Related work on requirements conflict analysis

The related work on the topic of requirements conflict analysis can be divided into two different categories: Formal and informal variants (also known as formal and qualitative reasoning).
Formal systems have specific knowledge about the contents of the requirements. They can analyze them to some extent and automatically check for conflicts to reduce human efforts.
Informal approaches are often based on negotiation between different stakeholders. They discuss the given set of requirements in a specific way and perform the conflict check either manually or semi-automated with tool support. In these approaches the tools can only help to guide and execute the negotiation process, but cannot find conflicts on their own.

Furthermore formal approaches can be divided into two more categories: Syntactic or semantic consistency.
Syntactic approaches try to find conflicts by using natural language processing techniques to check for keywords in the sentences of the requirements and compare them with each other.
Semantic approaches on the other hand do not only compare words, but try to compare the concepts behind the words with each other. This is especially important in today's projects, where lots of different people are involved. Even though they might share a common understanding of the project's concepts, the same concept might be labeled differently by several persons. This issue should not be an insurmountable obstacle for an approach.

Axel van Lamsweerde et. al. [Lamsweerde et. al., 1998] state that in general there is a lack of

- a precise and universal definition of what a conflict or inconsistency is.
- support for detecting conflicts. Current conflict management techniques presume that conflicts have been identified in some way.

- systematic techniques for resolving inconsistencies.

So for each approach it is important that it exactly defines the conditions that must be met by the requirements to result in a conflict. This definition does not have to be the same for all application contexts.

Some approaches that exist in the related work:

Hunter and Nuseibeh follow a logic based approach in order to manage inconsistent specifications [Hunter, Nuseibeh, 1997]. They state that in general all the needs of all stakeholders have to be elicited. But these stakeholders often under- or overspecify their requirements, which can result in inconsistencies. According to their work, inconsistencies are almost inevitable, but also have the advantage that they highlight areas that need more attention.
Hunter an Nuseibeh developed a logic called QC ("quasi-classical logic") to be able to formally describe requirements and find logical contradictions together with the usage of several analysis tools. They try to achieve scalability through the usage of so-called "ViewPoints", that cause that the analysis is only processed on parts of the specification with acceptable size.

Egyed and Grünbacher focus on the tracing between requirements to check for consistency [Egyed, Grünbacher, 2004]. Therefore they developed the "Trace Analyzer". This tool has to be provided with requirements that have been related to test scenarios. It can detect dependencies between different requirements if their test runs execute similar or overlapping lines of code, which means that the requirements affect the same part of the system, and indicates a potential conflict.
The downside to this approach is that executable code is necessary to identify potential conflicts, which is often not available in an early phase of the project. But early conflict analysis is very important to be able to reduce costs caused by inconsistent information.

Heitmeyer et al. describe a formal analysis technique, called "consistency checking", for the automated detection of specific kinds of errors [Heitmeyer et. al., 1995]. Examples are type errors, non-determinism, missing cases, and circular definitions. They use a tabular notation called SCR (Software Cost Reduction) for the requirements specification. Furthermore they introduce a formal semantics model based on a finite-state automaton to be able to analyze the requirements. The conducted two experiments conclude that tools for automated consistency check can be highly effective and reduce costs, assuming that explicit formal semantics are being used.

The downside of this approach is that it only considers syntactical consistency and does not address semantic conflicts.

3 Approach

The purpose of the approach presented in this master thesis is to combine the functionalities of project reporting with the capabilities of semantic technologies, especially ontologies.
The advantage of semantic techniques in the context of project reporting lies in the possibilities to enlarge the information value of the obtained data: Traditional systems are only able to deliver raw data without meaning to the user. My approach shall be able to interpret the data in a meaningful way and use this information to assist the user in the achievement of certain objectives.
Additionally, it shall be able to perform the most important general reporting tasks, like for example the presentation of data from different sources in a user-defined report (without semantic interpretation).

My approach is called “OntRep” (which is short for “Ontology Reporting”) and is being described in more detail in the following sections.

3.1 Architecture of OntRep

OntRep consists of different modules and is embedded into an environment called “Trac”. The following figure shows the coherences between the different modules:

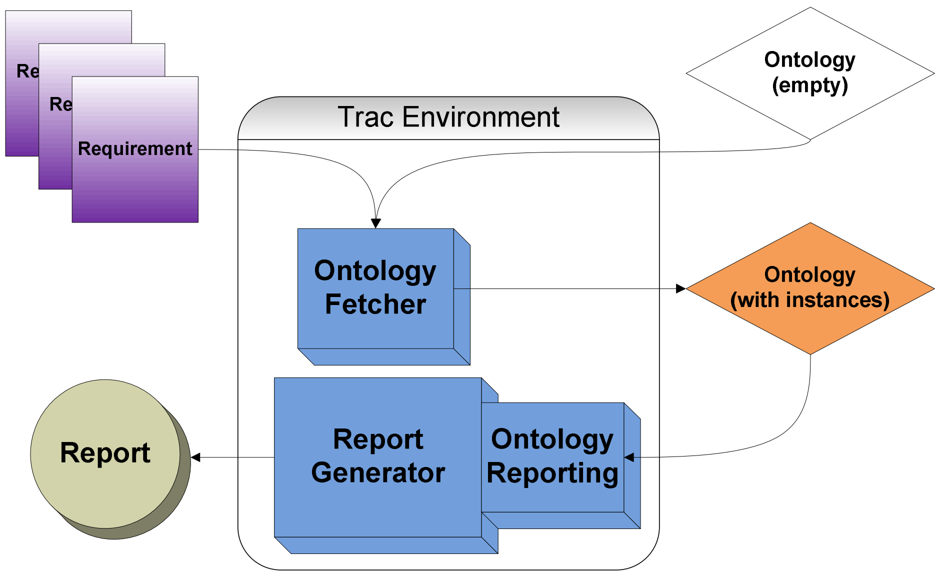
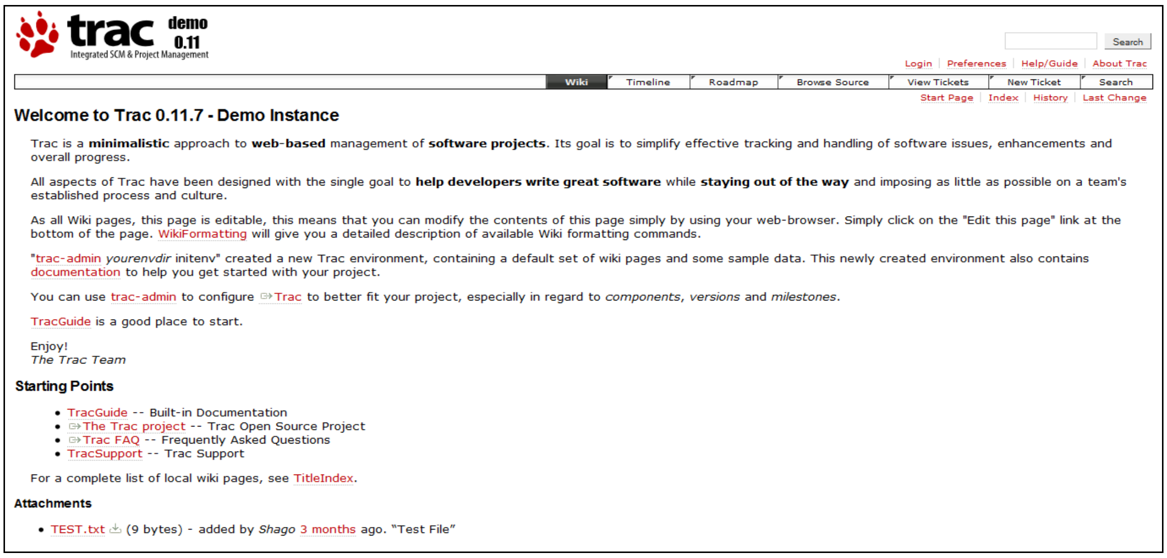


Figure 14: Architecture and coherences of the OntRep modules

OntRep builds upon a traditional reporting system. A traditional reporting system directly extracts the data from the defined data sources, like it can be seen in Figure 7 in section 1.2.
In my approach, I use a layer between the traditional reporting system and the data sources, that is responsible for semantic interpretation of data. This layer is being realized through the usage of an ontology. Figure 8 in section 1.2 shows where this layer is located.

3.1.1 The component-based architecture of OntRep

The prototype OntRep is realized within the web-based configuration and project management platform Trac [Trac]:


trac
demo
0.11
Integrated SCM & Project Management
Search
Login | Preferences | Help/Guide | About Trac
Wiki
Timeline
Roadmap
Browse Source
View Tickets
New Ticket
Search
Start Page | Index | History | Last Change

Welcome to Trac 0.11.7 - Demo Instance

Trac is a **minimalistic** approach to **web-based** management of **software projects**. Its goal is to simplify effective tracking and handling of software issues, enhancements and overall progress.

All aspects of Trac have been designed with the single goal to **help developers write great software** while **staying out of the way** and imposing as little as possible on a team's established process and culture.

As all Wiki pages, this page is editable, this means that you can modify the contents of this page simply by using your web-browser. Simply click on the "Edit this page" link at the bottom of the page. WikiFormatting will give you a detailed description of available Wiki formatting commands.

"trac-admin *yourenvdir* initenv" created a new Trac environment, containing a default set of wiki pages and some sample data. This newly created environment also contains documentation to help you get started with your project.

You can use trac-admin to configure Trac to better fit your project, especially in regard to *components*, *versions* and *milestones*.

TracGuide is a good place to start.

Enjoy!
The Trac Team

Starting Points

- TracGuide -- Built-in Documentation
- The Trac project -- Trac Open Source Project
- Trac FAQ -- Frequently Asked Questions
- TracSupport -- Trac Support

For a complete list of local wiki pages, see TitleIndex.

Attachments

- TEST.txt (9 bytes) - added by *Shago* 3 months ago. "Test File"

Figure 15: Trac start screen from a demo project

Trac is written in the programming language "Python". Its main features are a built-in wiki system for collaborative creation of text and a ticketing system for the organized management of tasks. Furthermore, a roadmap and a timeline provide an overview of the project tasks (called "tickets") in a chronological context.
Trac is characterized by a component-based architecture. This means that all functions in Trac are implemented through different modules, called components, built around the Trac core. When starting, the core searches for available components and loads them. This makes it possible that the functionality of Trac can be extended in a flexible and easy way through so-called "Plug-ins".

A plug-in is a ready-to-use package of one or more components, that work together to provide the user with certain functions. There are unlimited possibilities of use-cases for plug-ins. Examples are plug-ins that import data from a certain kinds of data source, allow new methods for authentication or offer project status visualizations on the start page.

I realized the prototype OntRep by making extensive usage of Trac's component-based architecture. Furthermore, the prototype itself takes advantage from this architecture by allowing to add other plug-ins to extend the prototype's functionality itself (like for example the possibility to install new drivers for data sources). Also see section 3.1.2 for more details.

OntRep consists of the following modules that work together to achieve its goals:

- **ReportGenerator**
  The ReportGenerator is responsible for providing the user with all standard functionalities that are necessary to perform general reporting tasks: For example access to data sources, designing report templates or generating reports from templates.
- **OntologyFetcher**
  The OntologyFetcher achieves the task of analyzing a given set of requirements and saving them correctly as instances in a given ontology.
- **OntologyReporting**
  The OntologyReporting module takes a prepared ontology as input, analyzes it and generates semantic reports based on the data, like for example the requirements conflict check.

Also see Figure 14, that shows the coherences of the three components the OntRep plug-in consists of.
The inputs for OntRep are a set of requirements and an empty ontology. The OntologyFetcher analyzes the given requirements and uses them to fill the ontology in a meaningful way, and delivers a new ontology (filled with instances) to the user. This ontology is taken as input for the OntologyReporting component, which analyzes the given ontology and generates the semantic reports in cooperation with the ReportGenerator.
The standard format the ReportGenerator produces are HTML reports, but it can be extended to also provide other formats.

All three components are part of the OntRep plug-in, but not all of these modules depend on each other: While the OntologyFetcher is independent from the other components, the OntologyReporting component can only be used together with the ReportGenerator. It is an extension for the ReportGenerator and makes use of some of its functionalities.

3.1.2 Extension points of OntRep’s component-based architecture

As already mentioned, the architecture of Trac supports the addition of new functionalities with ease through the usage of plug-ins. But they can only be “plugged” into certain points of the environment (meaning they can only provide new functionality at points, where it was intended to offer the possibility of plug-ins). These points are called “Extension points”. The Trac environment offers a wide range of possible extension points to be used. To use such an extension point, the plug-in has to implement a specific interface (meaning it has to implement the functions defined by the extension point).

Furthermore there is the option to equip the plug-ins themselves with extension points, so they can also be extended with new functionality.

Because my prototype makes plenty of usage of Trac’s component architecture and offers some extension points itself, the following diagram shall give an overview of the relations between the different components. The “lollipop” symbol marks an extension point, and an arrow the implementation of an extension point:

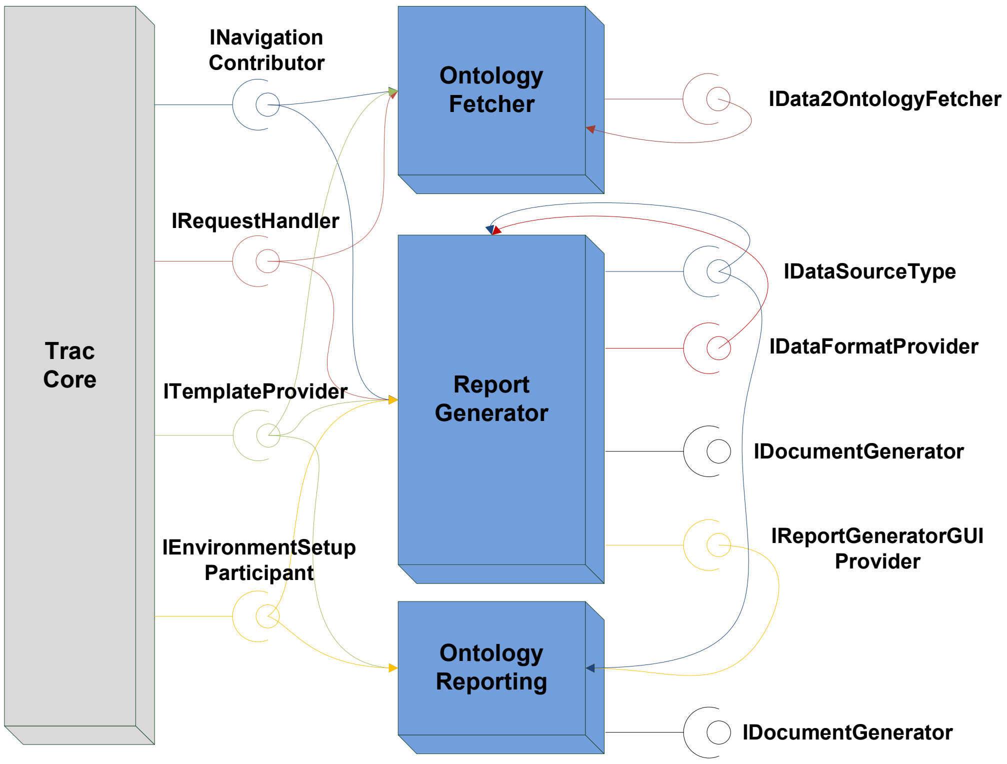


Figure 16: Overview of OntRep's provided and implemented extension points

The following extension points (and therefore the corresponding interfaces) from the **Trac core** are being implemented by OntRep:

- **INavigationContributor**
  This extension point allows a plug-in to add a new option into Trac's main navigation bar. The OntologyFetcher and the ReportGenerator each implement this interface and can be started from the navigation, as shown in the following screenshot:

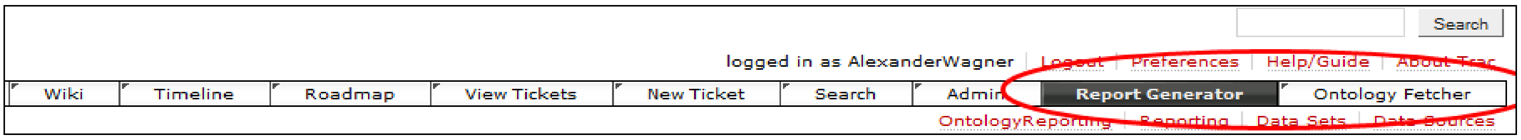


Figure 17: ReportGenerator and OntologyFetcher in Trac's navigation bar

- **IRequestHandler**
  This interface is the main extension point for a plug-in to be able to process the actions from a user. It delivers the HTTP requests sent by the user's browser to the plug-in.

This way the OntRep modules can react on button clicks and read the data provided by the user through HTML forms (like textfields or checkboxes).

It is implemented both by the ReportGenerator and the OntologyFetcher, because those two modules need input by the user to be operated. The OntologyReporting plug-in can also handle HTTP requests, but gets them indirectly via the ReportGenerator and not directly from the Trac core (also see the interface IReportGeneratorGUIProvider for more details).

- **ITemplateProvider**

  This extension point tells Trac that the plug-in provides its own HTML templates as user interface, and where to find them. This interface must be implemented by every plug-in that communicates with the user through HTML pages.

- **IEnvironmentSetupParticipant**

  Trac uses an internal database (by default SQLite, but PostgreSQL and MySQL are also supported) to save all data (like tickets, wiki pages, user accounts,…).

  A plug-in is also able to access this database if it needs a place to save information that shall be persistent. To keep the clear structure of the database, this information shall be saved in its own relational database table. By implementing this interface, a plug-in can tell the Trac environment that it needs to create its own tables in the database.

  The ReportGenerator creates some tables to save the information about the different data sources, queries, joined queries and report templates defined by the user. The OntologyReporting module also implements this interface to be able to create a table for the ontology reporting templates.

The following extension points are being defined and provided by OntRep itself. Most of them also have an implementation by OntRep:

- **IDataSourceType**

  This extension point is being offered by the ReportGenerator. By implementing this interface, a plug-in can allow the ReportGenerator to access a new kind of data source. It must define all necessary properties needed to access the data source type, and also the code to connect to such a data source and execute queries on it.

  The ReportGenerator does not only provide this extension point, it also offers two reference implementations (it is no problem that a plug-in implements its own extension point and can be quite useful to achieve a modular architecture):

One implementation allows the ReportGenerator to access PostgreSQL databases as a data source for reporting, the other one allows to access the internal Trac database. This way it is possible to generate reports based on the Trac data, for example about existing tickets or wiki pages.

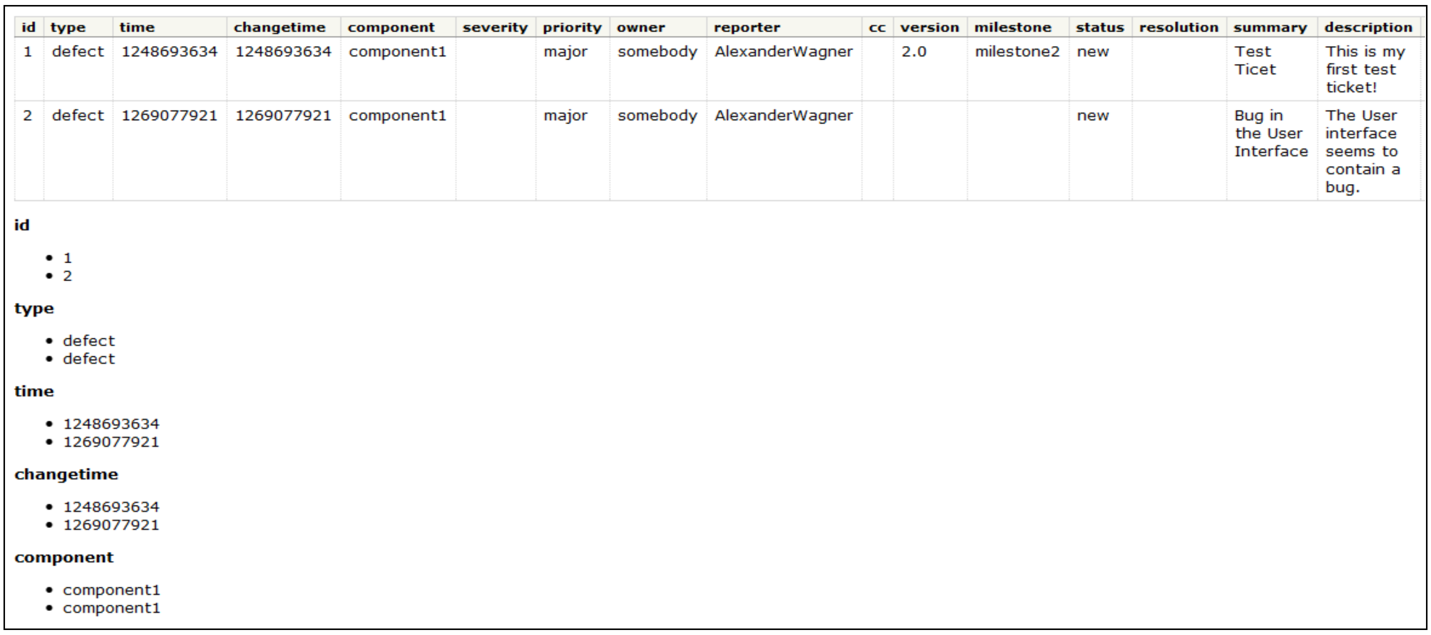
But other plug-ins can extend the supported kinds of data sources at will. This fact is being used by the OntologyReporting module, which adds “Ontology” as a new data source type. This way the user can manually extract data from an ontology by defining queries in the ontology query language SPARQL [SPARQL].

Note: This does not relate with the semantic categorization and conflict check of requirements. It is a way of manually extracting the exactly specified data from an ontology, which is an additional feature of OntRep.

- **IDataFormatProvider**

  This extension point allows to provide different formats of how the results of a query that has been executed on a data source will be rendered in a report. The implementers of this interface take the resulting data records as input and produce HTML code with them (because the base format of generated reports is HTML).

  The ReportGenerator provides two reference implementations: One formats the given data in a table, the other one produces a (numerical or bullet-points) list. The following screenshot shows an example:



id	type	time	changetime	component	severity	priority	owner	reporter	cc	version	milestone	status	resolution	summary	description
1	defect	1248693634	1248693634	component1		major	somebody	AlexanderWagner		2.0	milestone2	new		Test Ticet	This is my first test ticket!
2	defect	1269077921	1269077921	component1		major	somebody	AlexanderWagner				new		Bug in the User Interface	The User interface seems to contain a bug.

id

- 1
- 2

type

- defect
- defect

time

- 1248693634
- 1269077921

changetime

- 1248693634
- 1269077921

component

- component1
- component1

Figure 18: Query results in a generated report using the table and list format

- **IDocumentGenerator**

  This interface allows the user to download a generated report in a specific format. A plug-in implementing this extension point can take the generated HTML report as input, convert it into an arbitrary format and deliver it to the user. Example target formats are PDF, Word documents or Excel sheets. OntRep does not contain any implementing components by default.

  This extension point is offered both by the ReportGenerator and the OntologyReporting modules to be able to convert the generated reports into the possible formats.

- **IReportGeneratorGUIProvider**

  The ReportGenerator by itself allows to perform the most general reporting tasks (also see section 3.4.1 for more details). The user interface of this module is designed to serve these general purposes.

  If any plug-in intends to extend the reporting capabilities and therefore also needs a different or extended user interface, it can implement this extension point. It allows a plug-in to add a new option to the ReportGenerator's so-called "context navigation bar" and show its own user interface. HTTP requests sent to the ReportGenerator that aim at the extending plug-in will be looped through.

  This interface is implemented by the OntologyReporting module, which offers the user its own interface to create semantic reports. The following screenshot shows the corresponding entry in the context navigation bar:

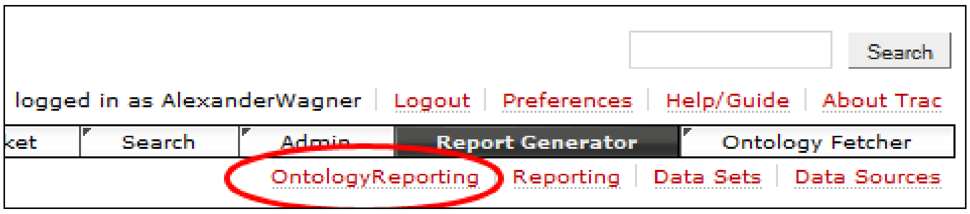


Figure 19: Adding a new entry to the ReportGenerator's context navigation bar

- **IData2OntologyFetcher**

  This extension point offered by the OntologyFetcher module allows to use plug-ins that fill an ontology with data. These plug-ins take the data records, an ontology and a list of specific parameters as input and create instances in the ontology based on the given data, and then return the filled ontology to the user.

OntRep provides two reference implementations for this interface: The **StandardTicketFetcher** takes a list of relational data and saves them in user-defined ontology classes. Each record becomes an individual in the ontology, and each column value is saved with an ontology property. There is no semantic interpretation of data.
More interesting is the second reference implementation, the **SemanticTicketFetcher**. This plug-in does analyze and interpret the contents of the given data, and considers this information when saving the individuals in the ontology. More details of how this component is working can be found in section 3.2.3.

The following sections describe different usage scenarios for OntRep. They describe the goals that shall be fulfilled by OntRep, how the user can achieve them and details of their technical functionality.

3.2 Usage Scenario 1: Categorization of requirements

The first usage scenario for OntRep describes how to automatically categorize software requirements with the prototype. More information on the advantages of categorized requirements can be found in section 1.4.

3.2.1 Goals for requirements categorization

- *The user shall be able to specify the categories the requirements shall be categorized into.*
  There should not be any limitations on the categories defined by the user, neither regarding their quantity nor their content.
- *OntRep shall be able to automatically relate a set of given requirements to the defined categories, based on syntactic and also semantic relationships.*
  The prototype shall not only check for matching keywords between the requirements and the categories, but also for conceptual relationships. This can be realized through natural language processing techniques, like checking for synonyms.
- *The user shall be able to manually specify additional synonyms for the defined categories.*

This goal means that there should be a possibility for the user to enhance the categories to improve categorization results and individualize the process.

It can be the case that there are certain terms used in a project, that are project-specific and do not belong to the standard English vocabulary (for example technical acronyms). The user shall be able to add such project-specific terms as additional user-defined synonyms to the knowledge base. There shall not be any limitations on these terms, neither regarding their quantity nor their content.

- *The user shall be able to manually negate synonyms for the defined categories.*

  This provides the user with the opportunity not only to add self-defined synonyms for the categories, but also to exclude certain terms as synonyms. For example, in many lexical databases “Safety” is a synonym to “Security”. But it can be the case that in the project-specific context these terms are different concepts, and shall not be handled as synonyms. OntRep shall provide the opportunity to negate such a relationship for the given project data. There shall not be any limitations on these terms, neither regarding their quantity nor their content.
- *OntRep shall be able to access the requirements data in a common format.*

  A common format for data interchange, like for example CSV or XML, shall make sure that the requirements data can be imported from a wide range of data sources.
- *The user shall be able to parameterize the behavior of the categorization process.*

  For the case the results produced by OntRep are not satisfying for the user, he shall be able to configure the behavior the prototype relates the requirements to the categories by defining a set of parameters.
- *The user shall be able to update the results of an already performed categorization process.*

  It might be the case that the requirements that were the input for an already performed categorization process have changed. OntRep shall provide the opportunity to update the requirements that exist in the semantic knowledge-base with the new, updated data.
- *OntRep shall separate the requirements that could not have been related from the rest.*

  If requirements could not have been related to any category by OntRep, they shall be saved in a special category (for example “Miscellaneous”).

3.2.2 Scenario description: Categorization of requirements

This section describes the steps the user has to perform to automatically categorize requirements with OntRep.

Prerequesites

The following prerequisites have to be met to be able to start the process of automatically saving requirements into semantically related ontology classes:

1. An empty ontology file (using the ontology language OWL-DL) exists.
2. The requirements exist in a *.csv file. Optionally, relations manually defined by the user in another *.csv file (for the case the user wants to set specific relations between requirements).
   The *.csv file with the requirements has to start with the column names in its first row (the names of the different attributes of each requirement). Then each record is described in each line, as shown in the following listing:

```
id,summary
1,The system shall provide the authenticated user the ability to see the network news at the News Portlet
2,The system will provide the user with a RSS Portlet
3,The system shall provide the user the ability to define the portlet layout
4,The system shall be able to clarify the interfaces to associated external systems
...
```

The (optional) user-defined relations have to be defined in a separate *.csv file. Each row contains the id of the requirement from which the relation originates, the id of the target requirement and type of relation. If no relation type is given the default type is called “references”. The following listing shows an example structure of this file:

```
2,27,consistsOf
2,13,partOf
23,2
21,30
30,23,consistsOf
...
```

Preparations

When all prerequisites have been met the user can start with the preparations necessary to start the fetching process:

1. The user first has to prepare an ontology, where he has to create several ontology classes that represent the categories the requirements will be related to. He has to create a root class, and one subclass for each category. Furthermore he must create an alternative class that will serve as container for all requirements that could not be related elsewhere.

   To solve this task I recommend the usage of the ontology editor "Protegé" [Protegé]. It is a free open source editor for comfortable processing and modeling of ontologies. A plug-in for Protegé allows to process the language OWL-DL.

   To create a class the user has to open the empty ontology file and select the subclass-explorer. An empty ontology does only contain the class "owl:Thing", which is the parent class of all other ontology classes. The user now has to select this class and click on the "Create subclass" button shown in the following screenshot:

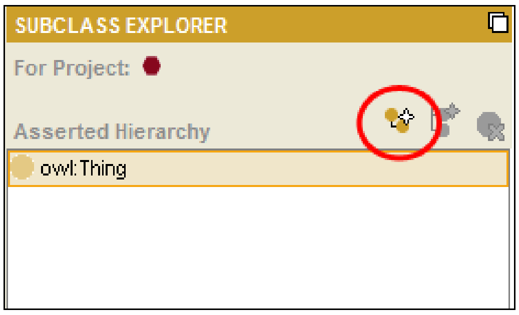


Figure 20: Creating a new ontology class in Protegé

   The user now has to create a root class (for example called "Requirement"). In the root class he must create an alternative class (for example "Miscellaneous") and one class for each category he wants to categorize the requirements into. The names of these category classes should be the category names. If the name of an intended category is too long or contains blank spaces, it can be defined using self-defined synonyms (see the next step for details).

   The following figure shows an example hierarchy with four categories defined in Protegé:

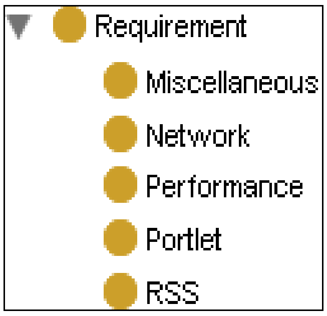
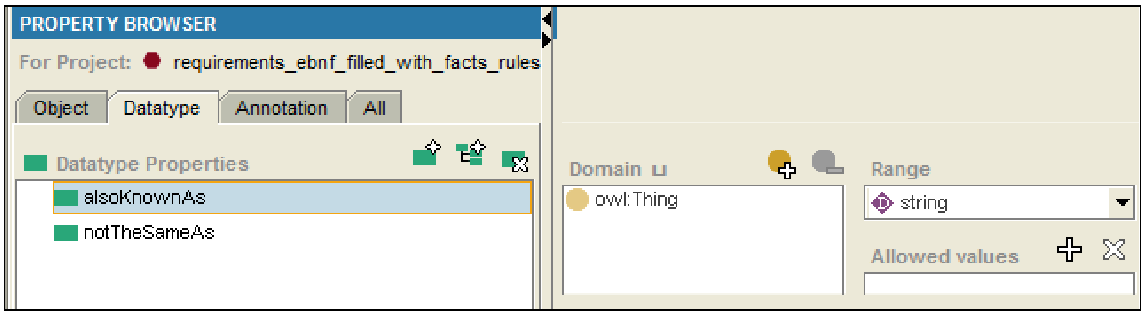


Figure 21: Example class hierarchy for categories in Protegé

2. To find the synonyms to the given categories, OntRep uses the lexical database WordNet (also see section 2.4). But it can be the case that the user wants to define some additional synonyms that shall be checked by OntRep. Furthermore, he might want to explicitly negate synonyms (defining that some words shall **not** be considered as synonyms, no matter if they are synonyms in the WordNet database).

   To specify his self-defined synonyms and self-defined negated synonyms, the user first has to declare the corresponding properties in the ontology (if they are not already existing). The user has to define two datatype properties (which are properties that relate to a value like a number or a string, and not to another individual) .

   They need to have the type string and the names “alsoKnownAs” and “notTheSameAs”. This can be done via Protegé in the property browser, as shown in the following screenshot:

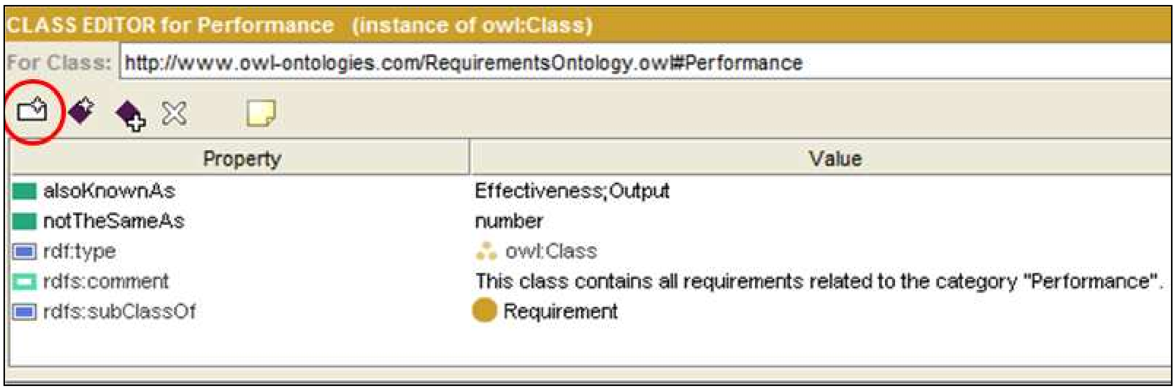


alsoKnownAs
notTheSameAs
owl:Thing
string

Figure 22: Defining the "alsoKnownAs" and "notTheSamAs" properties in Protegé

When the user has created the properties themselves, he can now specify the values for those properties for each category. To set these values the user has to select the desired class in the Protegé class browser and click the “Create datatype property value” button. Each property can be filled with multiple synonyms by using a semicolon as separator.

In the following screenshot the ontology class “Performance” has the synonyms “Effectiveness” and “Output”, but shall not be the same as “number” (like it is the case by default in WordNet):


CLASS EDITOR for Performance (instance of owl:Class)

For Class: http://www.owl-ontologies.com/RequirementsOntology.owl#Performance

Property	Value
alsoKnownAs	Effectiveness;Output
notTheSameAs	number
rdf:type	owl:Class
rdfs:comment	This class contains all requirements related to the category "Performance".
rdfs:subClassOf	Requirement

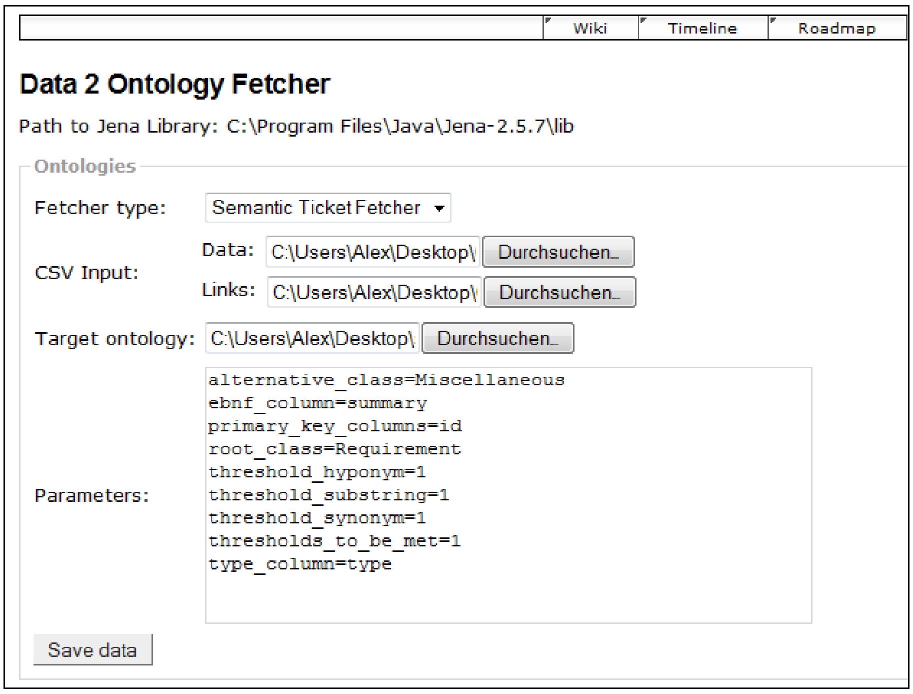
Figure 23: Specifying user-defined synonyms and negated synonyms

3. Optionally the user can add a comment to the ontology classes by using the “rdfs:comment” property and describe the category in more details (like shown in the previous screenshot). These comments can automatically be built into the reports to inform the report viewers about the categories.

Performing the fetching process

1. Now the user has to specify all necessary parameters that will influence the fetching process. This includes:
    - The path to the *.csv file that contains the requirements.
    - If available the path to the optional *.csv file that contains the user-defined relations between requirements.
    - The path to the *.owl ontology file that shall be filled with the requirements.
    - All necessary fetching parameters, like for example the name of the “root_class” (please see the technical description in section 3.2.3 for an accurate description of all parameters).

The following screenshot shows the interface where the user can specify all these parameters:



Data 2 Ontology Fetcher

Path to Jena Library: C:\Program Files\Java\Jena-2.5.7\lib

Ontologies

Fetcher type: Semantic Ticket Fetcher

CSV Input: Data: C:\Users\Alex\Desktop\ Links: C:\Users\Alex\Desktop\

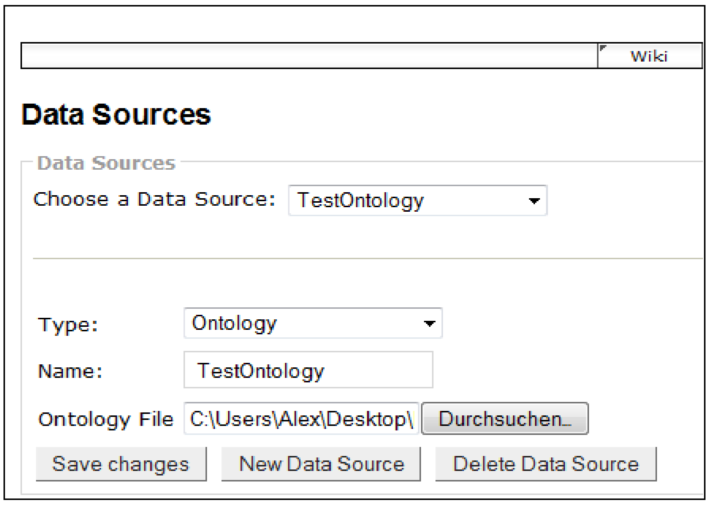
Target ontology: C:\Users\Alex\Desktop\

Parameters:

```
alternative_class=Miscellaneous
ebnf_column=summary
primary_key_columns=id
root_class=Requirement
threshold_hyponym=1
threshold_substring=1
threshold_synonym=1
thresholds_to_be_met=1
type_column=type
```

Figure 24: Configuring the parameters for the fetching process

2. Now the user can click the button “Save data” to initiate the fetching process. Depending on the number of requirements, this process can take several minutes to finish.
3. When finished the user has to save the filled ontology file. He now has to configure it as a data source for the ReportGenerator.
   To do so the user has to specify the file as a data source with type “ontology” and a self-given name. This has to be done the menu “data sources”, shown in the following screenshot:



Data Sources

Data Sources

Choose a Data Source: TestOntology

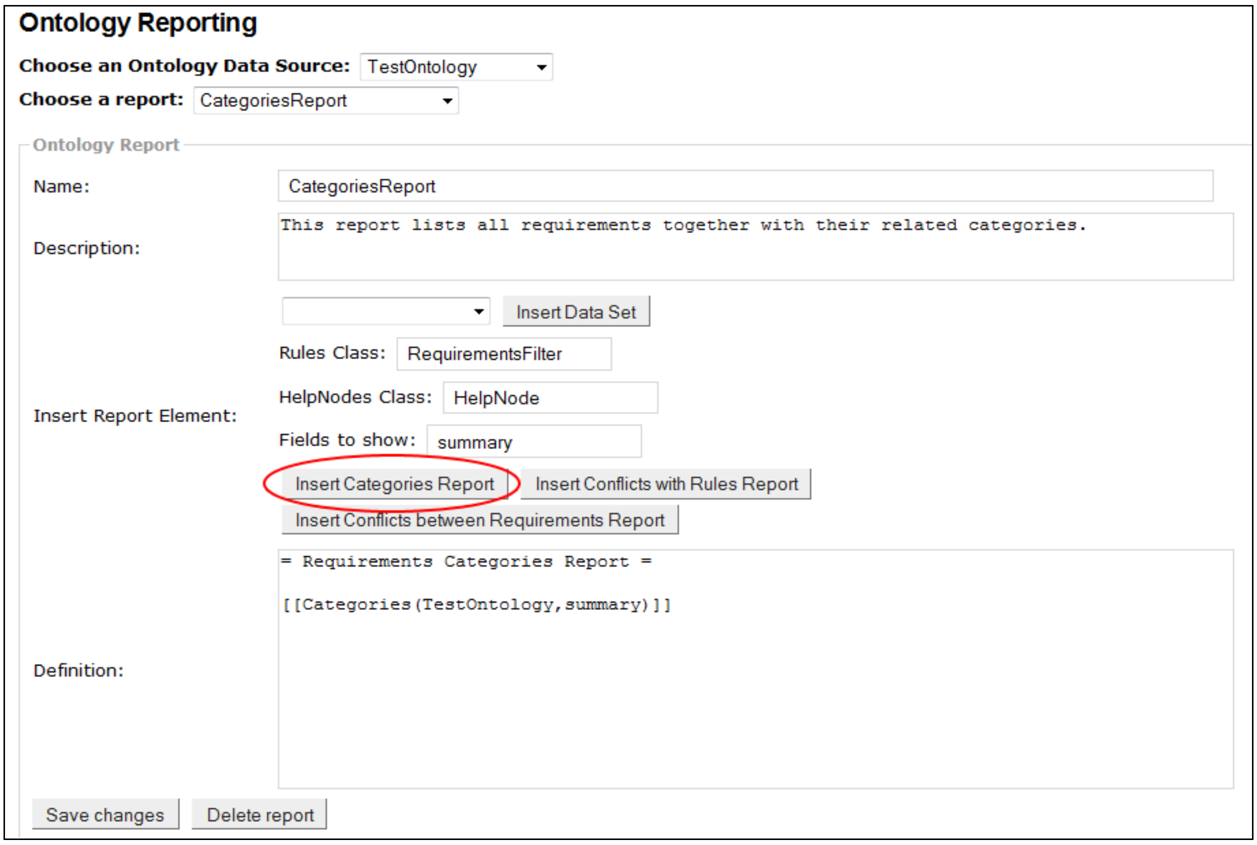
Type: Ontology

Name: TestOntology

Ontology File C:\Users\Alex\Desktop\ Durchsuchen...

Figure 25: Saving the filled ontology as a data source for the ReportGenerator

4. Now the user can create a report by using the whole Trac wiki formatting syntax. There he can insert an element that automatically creates a listing of the requirements and their related categories based on the data in the ontology. To do so, he has to click the button "Insert Categories Report" in the OntologyReporting interface, like shown in the following screenshot:



Ontology Reporting

Choose an Ontology Data Source: TestOntology

Choose a report: CategoriesReport

Ontology Report

Name: CategoriesReport

Description: This report lists all requirements together with their related categories.

Insert Report Element:

Rules Class: RequirementsFilter

HelpNodes Class: HelpNode

Fields to show: summary

Insert Data Set
Insert Categories Report
Insert Conflicts with Rules Report
Insert Conflicts between Requirements Report

Definition:

```
= Requirements Categories Report =

[[Categories(TestOntology,summary)]]
```

Figure 26: Creating the requirements categorization report

5. When the user has finished editing the report template he can now see a preview of the generated report, or download it in different formats (depending on the format plug-ins that are installed). The following screenshots show the buttons to initiate the report generation process and an example of a generated report:

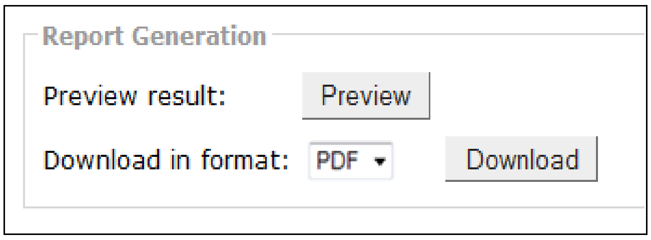
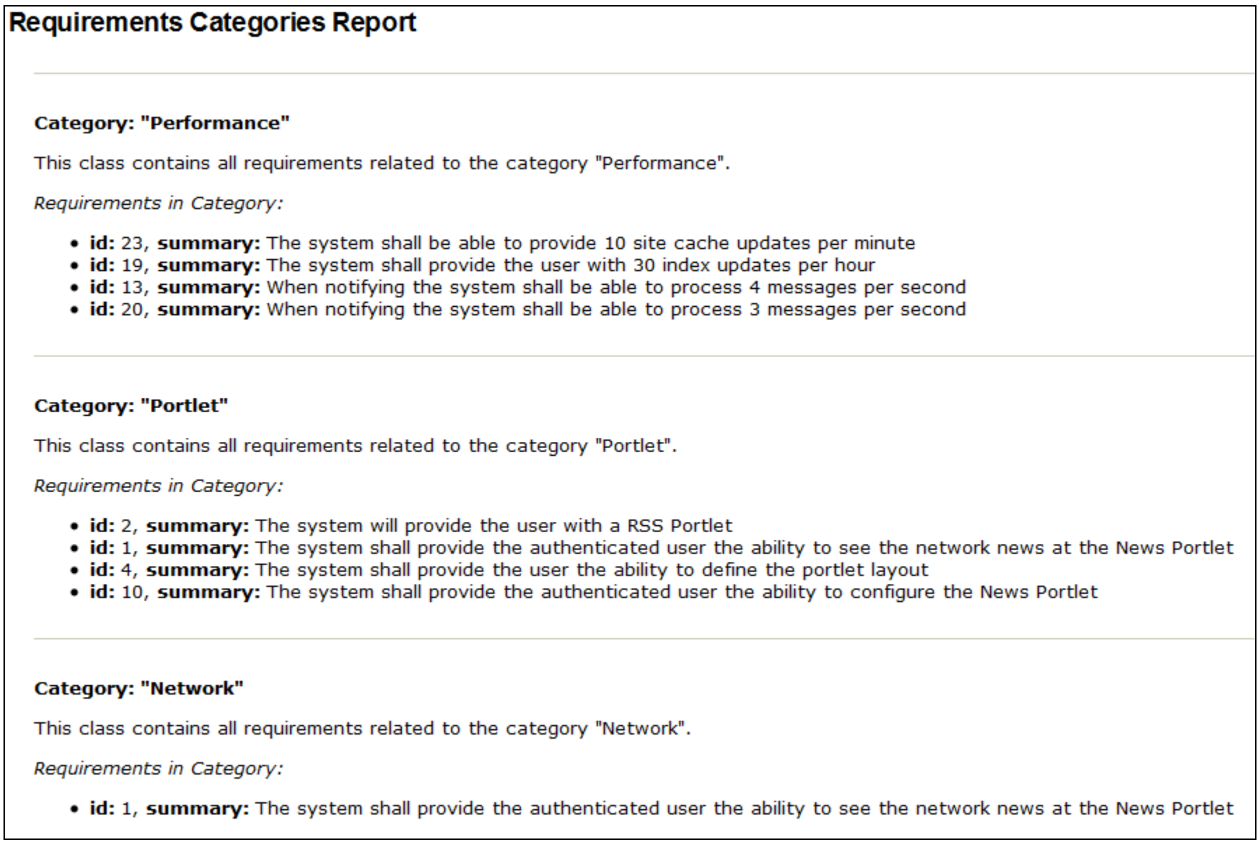


Figure 27: Generating the report



Requirements Categories Report

Category: "Performance"

This class contains all requirements related to the category "Performance".

Requirements in Category:

- **id:** 23, **summary:** The system shall be able to provide 10 site cache updates per minute
- **id:** 19, **summary:** The system shall provide the user with 30 index updates per hour
- **id:** 13, **summary:** When notifying the system shall be able to process 4 messages per second
- **id:** 20, **summary:** When notifying the system shall be able to process 3 messages per second

Category: "Portlet"

This class contains all requirements related to the category "Portlet".

Requirements in Category:

- **id:** 2, **summary:** The system will provide the user with a RSS Portlet
- **id:** 1, **summary:** The system shall provide the authenticated user the ability to see the network news at the News Portlet
- **id:** 4, **summary:** The system shall provide the user the ability to define the portlet layout
- **id:** 10, **summary:** The system shall provide the authenticated user the ability to configure the News Portlet

Category: "Network"

This class contains all requirements related to the category "Network".

Requirements in Category:

- **id:** 1, **summary:** The system shall provide the authenticated user the ability to see the network news at the News Portlet

Figure 28: Example categorization report

3.2.3 Technical description: Categorization of requirements

After I described how the user has to operate OntRep to automatically categorize requirements (the external view of the categorization process), I will now technically describe how it performs the categorization of requirements (the internal view of the categorization process):

Start of fetching process

If all prerequisites described in section 3.2.2 are met, OntRep can start the fetching process:

1. The OntologyFetcher plug-in loads the ontology (usually in form of an *.owl file) and the requirements (provided in form of a *.csv file) which have been submitted by the user. Optionally, the user-defined relations between the requirements are also loaded if available (also in form of a *.csv file).
2. The OntologyFetcher reads in the values of the following parameters provided by the user:
    - `alternative_class`: This parameter specifies the name of the alternative ontology class. It shall serve as the alternative target for requirements that could not be related with any other class by the OntologyFetcher (default value: “Miscellaneous”).
    - `ebnf_column`: This parameter specifies which column of the requirement records has been defined using the EBNF grammar (described in more detail in section 3.3.3) and shall be analyzed (only relevant for conflict check of requirements, default value: “summary”).
    - `primary_key_columns`: This parameter specifies which column(s) make up the primary key of each requirement. By defining multiple columns the primary key is composed of all the columns together.
      This is necessary to be able to recognize duplicate requirements (default value: “id”).

- `root_class`: This parameter specifies which ontology class contains the subclasses that serve as categories for the requirements. These subclasses will be filled with the requirement's data (default value: "Requirement").
- `threshold_hyponym`: This threshold defines the number of hyponyms (subclass-words, e.g. "dog" is a hyponym of "animal") that have to be found in the text of a requirement to meet this threshold (also see the next section for more details, default value: 1).
- `threshold_substring`: This threshold defines the number of substring relationships (partial words, e.g. "project" and "projectmanager") that have to be found in the text of a requirement to meet this threshold (also see the next section for more details, default value: 1).
- `threshold_synonym`: This threshold defines the number of synonyms (different words with the same meaning) that have to be found in the text of a requirement to meet this threshold (also see the next section for more details, default value: 1).
- `thresholds_to_be_met`: This parameter defines the number of thresholds that have to be met in total to relate a requirement to a category class. Only a value between 1 and 3 makes sense (also see the next section for more details, default value: 1).
- `type_column`: This parameter is not being used by the SemanticTicketFetcher. It can be used by other fetcher types (and is being used by the StandardTicketFetcher, see section 3.1.2). It allows to relate requirements to categories depending on the value of the defined column, without any semantic interpretation of the contents (default value: "type").

3. When the user has provided all necessary the inputs, the actual fetching process starts after a click on the "Save data" button. The OntologyFetcher now iterates through all requirement records loaded from the corresponding *.csv file.
4. The OntologyFetcher checks the primary keys of each loaded requirement and removes all requirements with the same keys that may already exist in the ontology. This has to be done to make sure there will be no duplicate requirements. This approach also allows to update an ontology that has already been filled with requirements with the newest versions of these requirements.

Another important detail is the way the primary keys are saved in the ontology. This is being done by using a so-called “rdf:Bag” data structure. It allows to relate composite values to an individual. It is necessary in this case, because the primary key of a requirement could be consisting of more than one attribute. The following figure shows an example of how an rdf:Bag can be used to map a composite primary key to a requirement. The primary key in this example consists of the values for “serial” and “date”:

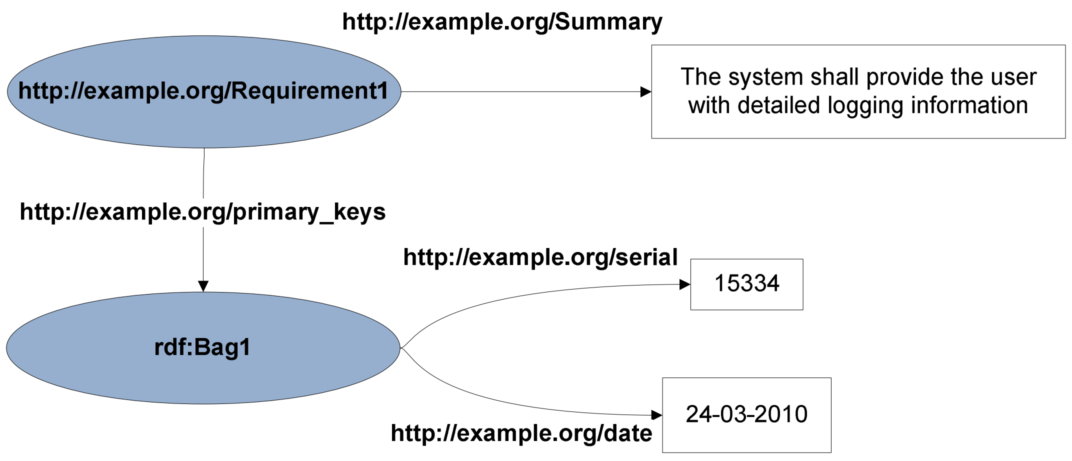

http://example.org/Summary
http://example.org/Requirement1
The system shall provide the user with detailed logging information
http://example.org/primary_keys
http://example.org/serial
15334
rdf:Bag1
24-03-2010
http://example.org/date

Figure 29: Mapping a composite primary key to a requirement using a rdf:Bag

5. Now the OntologyFetcher iterates through all ontology classes that are subclasses of the specified root class (i.e. the classes that represent the categories).
   For each of those classes it checks for user-defined synonyms and user-defined negated synonyms related to the class (like described in section 3.2.2), and loads them into a list in the memory.

Relating the requirements

Now all necessary preparations have been done to start the **check for relation between the requirement and a category:**

1. At first all “stopp-words” are being removed from the words in the requirement. These words have low informational relevance and therefore won’t be taken into account. The following listing shows the stopp-words that will be removed [Stopp-words]:

```
"a","able","about","across","after","all","almost","also","am","among","an","and",
"any","are","as","at","be","because","been","but","by","can","cannot","could","dear",
"did","do","does","either","else","ever","every","for","from","get","got","had",
"has","have","he","her","hers","him","his","how","however","i","if","in","into","is",
"it","its","just","least","let","like","likely","may","me","might","most","must",
"my","neither","no","nor","not","of","off","often","on","only","or","other","our",
"own","rather","said","say","says","she","should","since","so","some","than","that",
"the","their","them","then","there","these","they","this","tis","to","too","twas",
"us","wants","was","we","were","what","when","where","which","while","who","whom",
"why","will","with","would","yet","you","your"
```

2. Then all remaining words of the requirement are being brought to a comparable root form. This process is called “Stemming”, and is done with a well-known algorithm, the “Porter-Stemmer” algorithm [Porter-Stemmer].
   An example is to stem “jumping” to “jump”, or “logical” to “logic”.
   For our purposes, the exact grammatical form of a word is not important. After stemming, the OntologyFetcher can compare the words no matter what form they have. The same word in two different forms will always be stemmed to the same root form.
3. Now the fetcher gets all synonyms and hyponyms of all the words of the category (i.e. of the name of the ontology class combined with all the user-defined synonyms) by using the natural language processing library “WordNet”.
   Synonyms are different words that have the same or similar semantic meaning (e.g. “student”, “pupil” and “scholar”). A hyponym is a sub-word, a word that describes a subcategory of another word. For example “dog” is a hyponym to “animal”, and “painting” is a hyponym to “art”. User-defined negated synonyms are not taken into account as synonyms, no matter if they are WordNet synonyms.
   Using these associations it is possible to find semantic relationships between the requirements and the categories that go beyond key-word matching.
4. Then all substring relationships between the category-words and the requirement words are being checked, meaning the OntologyFetcher checks if there are any words contained by others. For example “net” is a substring of “network”. This way a relationship between “project” and “projectmanagement” can be found for example.
5. Now the number of all matches (hyponym-, synonym- and substring matches) are being delivered to a function that decides whether or not there is a relationship between the given requirement and the given category.
   This function takes the following inputs:

- The necessary thresholds for the hits of synonym, hyponym and substring matches (defined with the parameters “threshold_synonym”, “threshold_hyponym” and “threshold_substring” by the user).
- The actual number of hits for synonym, hyponym and substring matches.
- The number of thresholds that have to be met in order for a positive result (defined with the parameter “thresholds_to_be_met” by the user).

The function checks if the actual hits for synonym, hyponym and substring matches meet the necessary threshold values. So the actual number of thresholds met is always between zero and three. If this number is equal or higher than the number of “thresholds_to_be_met”, the requirement will be related to the category, otherwise not.

Saving the requirements

If the OntologyFetcher detects a relation between the requirement and the category, **it saves the requirement in the corresponding ontology class**:

1. At first, the OntologyFetcher creates an empty individual in the ontology class that has been identified as related to the requirement. This individual is anonymous, meaning it cannot be referenced through an unique URL.
2. The OntologyFetcher now iterates through all columns of the requirement’s record (like for example “id”, “summary”, “description”,...), creates a literal for each one of them and relates them with the individual. These datatype relations are being realized through properties named after the column names (e.g. “id”, “summary”, “description”,...). If such a property does not already exist in the ontology, it is being created by the OntologyFetcher.
   Furthermore, the column(s) declared as primary key (by using the parameter “primary_key_columns” defined by the user) are being saved in a newly created “rdf:Bag” like shown in Figure 29.
3. The OntologyFetcher now checks if the element has already been saved in another class as well, by checking for individuals with the same primary key. This is the case if a requirement is related to more than one category.

The OntologyFetcher declares that the new individual is the same as the already existing one by using the “owl:sameAs” property. This property is already predefined by the OWL language and means that the two individuals related through the property actually describe the same thing. In this case it has the semantics that “the two individuals in two different categories actually represent the same requirement”.

4. If the requirement could not have been related to any category, it is being saved in the user-defined “alternative class”. This is the case if the defined thresholds have not been met for any category class.
5. If the user provided self-defined relationships via the optional *.csv file, those references are being built now, by reading the primary key of the start and the target requirement, and the name of the desired relation. Relationships without any defined name get the standard name “references”.
6. The filled ontology file is returned to the user, who can download it now.
7. The OntologyReporting plug-in is now able to create a requirements categorization report from the filled ontology. Therefore it loads all subclasses of the defined root-class (e.g. “Requirement”), and reads all individuals in those classes. The results are being transformed into HTML, which is provided to the user.

3.3 Usage Scenario 2: Requirements conflict analysis

The second usage scenario for OntRep describes how to automatically check software requirements for certain types of conflicts. More information on the problems that arise with inconsistent requirements can be found in section 1.4.

3.3.1 Goals for requirements conflict analysis

- *OntRep shall be able to automatically recognize an accurately defined set of conflict types based on the semantic content of requirements.*
  Before it is possible to find any kind of conflict in a given set of requirements, it needs to be defined what a conflict exactly is in our context. The prototype shall be able to check for certain types of conflicts that are likely to happen in software development projects. Also see section 3.3.2 for an exact definition of conflict types that shall be found.

To perform this conflict check, OntRep must be able to semantically understand the contents of the requirements to a certain extent.

- *The user shall be able to define facts that expand the knowledge about the system to develop.*

  This will be realized through the use of semantic technologies.

- *The user shall be able to group different objects from the contents of the requirements that share commonalities.*

  A common use for this goal is the definition of terms for a glossary. In a glossary, certain terms that have been used are being defined in more detail. For my prototype, these glossary terms serve the purpose to group the objects that are being described in the requirements.

  For example one requirement could describe how the user accesses an “administration panel”, and a second requirement the access to the “database”. Now the terms “administration panel” and “database” could be grouped together as “Secured Resource”, that shall be handled in a special way. This grouping will be realized through the use of semantic technologies.

- *It shall be easy to modify existing knowledge about the system and add new knowledge.*

  Because requirements are often not very stable in projects (they often change during the project) it is important that facts about the systems can be altered afterwards.

- *OntRep shall be able to infer new knowledge out of existing facts and consider it during the conflict analysis.*

  This process is called “Reasoning” and will be realized with semantic technologies. Also see section 1.3.2 for more information on reasoning.

3.3.2 Definition of conflict types that shall be recognized

The prototype shall be able to recognize the following types of conflicts that may occur within the requirements or the constraints they underlie.

This list of conflict types makes no claim to be complete. Many other types of conflicts may occur in a certain project context. It can also be the case that the conflict types that are being described here do not occur in a certain project at all. It is a reasonable selection that’s appropriate in respect to the implementation efforts for this master thesis and the benefits it might bring:

- **Conflict type “CRC”: Conflicts between requirements and business constraints**
  Business constraints are external requirements, thus requirements that must not be altered and have higher priority than the software requirements, and constrain their content. If any software requirement makes a statement that contradicts a business constraint, there is a conflict of type “CRC”.

  Example:
  - *Requirement 1*: The system shall provide the user with access to an administration panel.
  - *Business constraint 1*: Only “trusted persons” shall be able to access “secured resources”.
  - *Fact 1*: The administrator is a “trusted person“.
  - *Fact 2*: The “administration panel” is a “secured resource“.

  → *Conflict*: The user is not a “trusted person”! (if there are no other facts in the knowledge base, then the administrator is the only “trusted person”)

- **Conflict type “CRG”: Conflicts between requirements and documentation guidelines**
  It shall not only be possible to constrain the requirements with regard to content, but also with regard to structure. Documentation guidelines shall make sure that the requirements meet certain formal conditions and assert a level of precision within all requirements. If any requirement does not meet the constraints specified in a documentation guideline, there is a conflict of type “CRG”.

  Example:
  - *Documentation Guideline 1*: Each requirement must specify a role it applies to.
  - *Requirement 1*: The system shall provide access to the database.
  - *Requirement 2*: The system shall provide the tester with access to the IDE.
  - *Requirement 3*: The system shall provide statistics and logging information.

  → *Conflict*: Requirement 1 does not specify a role it applies to!
  → *Conflict*: Requirement 3 does not specify a role it applies to!

- **Conflict type “CRR”: Numerical conflicts between different requirements**

  In a software requirements specification with a large number of requirements, usually there are many statements with numerical specifications (for example exact performance demands).

  It can be the case that different requirements make different numerical statements about the same thing. This results in a potential conflict, that shall be recognized by the prototype and presented to the user. The user then must decide if it is a real conflict in his project context.

  Furthermore, these types of conflict can involve technical constraints. These are constraints that are caused by the technological context and can’t be altered. A typical constraint of type “CRR” occurs when one requirement intends to use a technology that is in conflict with another requirement due to technical constraints.

  Example 1:

  - *Requirement 1*: The user shall be able to send 50 notification messages per hour.
  - *Requirement 2*: The user shall be able to send 60 notification messages per hour.

  → *Potential conflict*: The two requirements make different numeric statements for the same thing!

  Example 2:

  - *Requirement 1*: The user shall be able to send 50 notification messages per hour.
  - *Requirement 2*: The system shall send notification messages via RSS.
  - *Technical constraint 1*: The RSS module that’s being used allows to send 40 notification messages per hour on average.

  → *Potential conflict*: The two requirements imply different numeric statements for the same thing!

3.3.3 Scenario description: Requirements conflict analysis

This section describes the steps the user has to perform with OntRep to automatically check requirements for potential conflicts.

Prerequisites

The process of automatically checking requirements for the three conflict types (which are described in section 3.3.2) has similar prerequisites as the automatic categorization of requirements, but some additional demands:

1. For automatic conflict analysis, OntRep has to analyze the semantic contents of each requirement. Because it would be quite difficult to analyze any kind of sentence with today's natural language processing techniques, I decided to constrict the allowed structure for requirement's content. This allows a much higher precision regarding the conflict check, without losing too much flexibility in phrasing the requirements.
   To be able to analyze the requirements semantically, I defined a grammar. The text of the requirement's content has to follow this grammar. I specified the grammar by using the meta-language "Extended Backus–Naur Form" [EBNF].
   The grammar is based on an already existing requirements grammar in literature [Rupp, 2004], but has been modified and extended by myself. The following listing shows the definition of the grammar:

```
<requirement> := <constraints> "THE SYSTEM" <obligation> <type_phrase>

<constraints> := (<when> | <under_what_conditions>)
<when>        := *temporal condition under which the requirement is valid*
<under_what_conditions>  := *logical condition under which the requirement is valid*

<obligation>  := ("SHALL" | "SHOULD" | "WILL")

<type_phrase> := (<process_details> [<thing_to_be_processed>] | "PROVIDE" <whom>
<stop_word> <thing_to_be_processed>)

<process_details> := (<process_verb> | "PROVIDE" <whom> "THE ABILITY TO"
<process_verb> | "BE ABLE TO" <process_verb>)
<process_verb> := (<one_word_process_verb> | <muliple_word_process_verb> <stop word>)
<one_word_process_verb> := *verb, consisting of one word, which characterizes the
functionality*
<multiple_word_process_verb> := *verb, consisting of more than one word, which
characterizes the functionality*

<whom> := *Person that is provided with the functionality*

<stop_word> := *english stop word*

<thing_to_be_processed> := *Object of the processing*
```

The grammar has to be read from top to bottom. Each line defines a production rule, a rule describing how to decompose the term on the left side. The terms on the left side of the rules are so-called “nonterminal symbols”, meaning that they can be decomposed into other elements (they are marked in this grammar with the angle brackets “<” and ”>”). The right side of the production rules define into what elements the nonterminal symbols can be decomposed. This can also include other nonterminal symbols. The grammar shown above furthermore makes usage of the following elements:

- A constant text in capital letters, e.g. “THE SYSTEM”.
- An alternative choice from some given options, e.g. (“OPTION 1” | “OPTION 2” | “OPTION 3”).

- An optional element which can be included in the production or omitted, e.g. [“OPTIONAL”].
- Free text that describes the symbol in more detail, e.g. *Person that is provided with the functionality*.

The grammar is structured as follows:
Because it describes requirements, the first nonterminal symbol is “requirement”. They are being structured into a conditional part (either logical or temporal conditions), then the constant “THE SYSTEM”, an obligation word (“SHALL”, “SHOULD” or “WILL”) and a “type_phrase”.
Latter can be decomposed in different ways, but the most important elements are “whom” (describing a person or role), the “process_verb” (describing the action that is being performed in a verb) and the “thing_to_be_processed” (describing the thing on which the action is being performed).
English stopp-words (also see section 3.2.3) are being used as a natural form of delimiter between different elements. This approach allows that requirements can be formulated in a comparatively natural way, without losing much precision. Here is a list of example requirements that can be built with the specified grammar:

- *The system shall use SSL encryption.*
- *The system shall be able to process 3 messages per second.*
- *The system shall provide anybody with boot information.*
- *When booting the system shall provide the user the ability to access the sprint definition panel.*
- *The system shall provide the authenticated user the ability to see web-statistics.*
- ...

It has to be noted here, that in the best case the requirements have been defined in the specified grammar from the start, so no conversion is necessary. This goal can be achieved much more likely through the usage of requirements management tools that support using a requirements grammar.

2. Beside the prerequisite that the requirements have to follow the specified EBNF grammar, the ontology file has to be prepared with some classes that will serve as the

container for the different parts of the analyzed requirements. These classes are needed additionally to the categorization classes (that were described in section 3.2.2). The following classes are needed:

- EBNF_TemplateElement: This class is the root class to all of the following classes.
- Condition: This class saves the “condition” part of the requirements.
- Obligation: For the “obligation” part of the requirements.
- Person: For the “whom” part of the requirements.
- Process: For the “process_verb” part of the requirements.
- ThingToBeProcessed: For the “thing_to_be_processed” part of the requirements.
- HelpNode. This class saves so-called “help nodes”, that are necessary for the modeling of some facts.

It is useful to have a reusable, empty ontology template already containing these classes. The following screenshot shows their structure in Protegé:

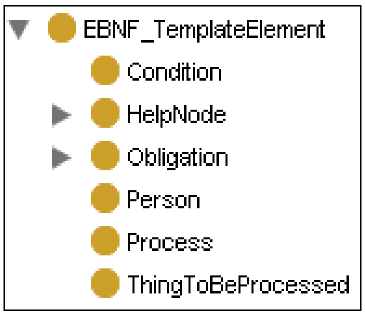


Figure 30: The class hierarchy for the different parts of a requirement

Preparations

1. After the user has prepared the ontology file with all necessary classes, and converted the requirements into the specified grammar, he can import them into the ontology. This can be done with the OntologyFetcher by performing the same procedure as described in section 3.2.2. Mind to set the parameter named “ebnf_column”: It must be set to the column in the requirement records that contains the content in EBNF grammar, and shall therefore be analyzed.

When finished the subclasses of the “EBNF_TemplateElement” class have been filled with new individuals (additionally to filling the requirement’s categories).
The class “Person” now contains all persons or roles mentioned in the requirements, the class “ThingToBeProcessed” contains all things acted upon in the requirements, and so on.

2. Now the user has to define all facts in the ontology that are relevant for him in his context. Modeling facts in an ontology is a very broad topic, and there are countless ways to do it. There are no limitations except for the technological and semantic capabilities of OWL. I will cover the most relevant examples here regarding the check for the specified conflict types (see section 3.3.2):

- It is often necessary to group different “things to be processed” from the requirements together and make a class of “things to be processed” that share something in common. In our context of requirements engineering, this approach can be done to model “glossary terms”, which define the names used in the project in more detail.
  An example is to define which things that have been covered by the requirements shall be “secured resources” (with the semantic of being resources that shall not be accessible by anybody).
  To specify this, the user can create an ontology class and list all things that shall be “secured resources” as an so-called “enumeration” (meaning that each element that shall be a member of the class will be listed manually). This has to be done in the “necessary & sufficient” part of the class definition in “Protegé”.
  All individuals in the ontology that meet a “necessary & sufficient” definition (which is a so-called “restriction”) are automatically computed as members of the class by the reasoner. So the specified “things to be processed” are recognized as instances of the corresponding class (no matter what other classes they might be in additionally).
  The following screenshots show the class hierarchy of some example glossary terms, and the “necessary & sufficient” definition of the class “SecuredResource” in Protegé:

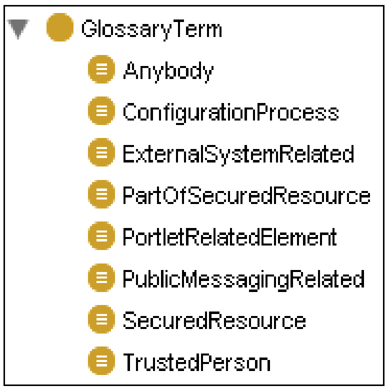

GlossaryTerm
Anybody
ConfigurationProcess
ExternalSystemRelated
PartOfSecuredResource
PortletRelatedElement
PublicMessagingRelated
SecuredResource
TrustedPerson

Figure 31: Class hierarchy for some glossary terms

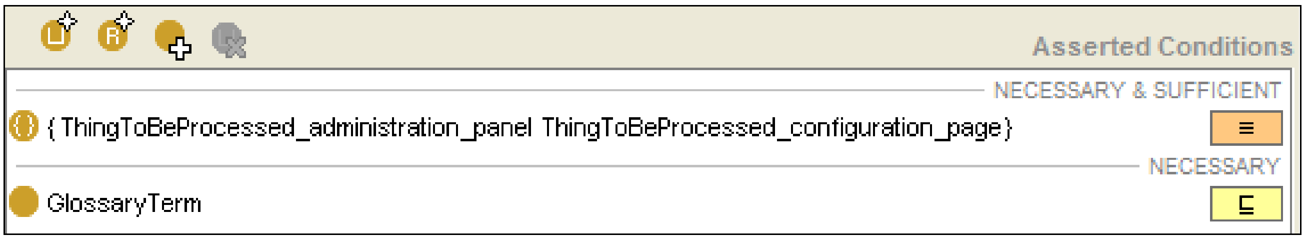

Asserted Conditions
NECESSARY & SUFFICIENT
{ThingToBeProcessed_administration_panel ThingToBeProcessed_configuration_page} ≡
NECESSARY
GlossaryTerm ⊑

Figure 32: Definition of the "administration panel" and the "configuration page" as "secured resources"

- The user can also model numerical statements about the "things to be processed" from the requirements. A requirement could say that "The system has to use SSL encryption". Now the user could state that "SSL encryption allows to process 3 messages per second". This fact would represent a technical constraint.

  To do so, he has to create a help node and an individual that represents the object "messages per second". He must now connect the individual "SSL encryption" with the numerical value "3" and the object "messages per second" by using the help node. The properties he has to use are "thing_to_be_processed" and "hasNumericValue". The following figure shows the example graph:

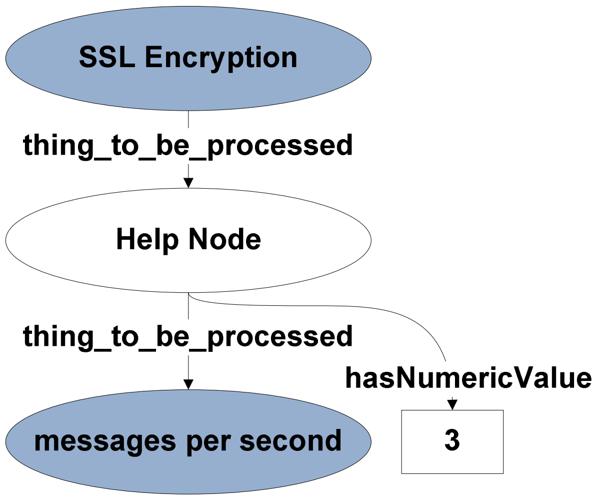
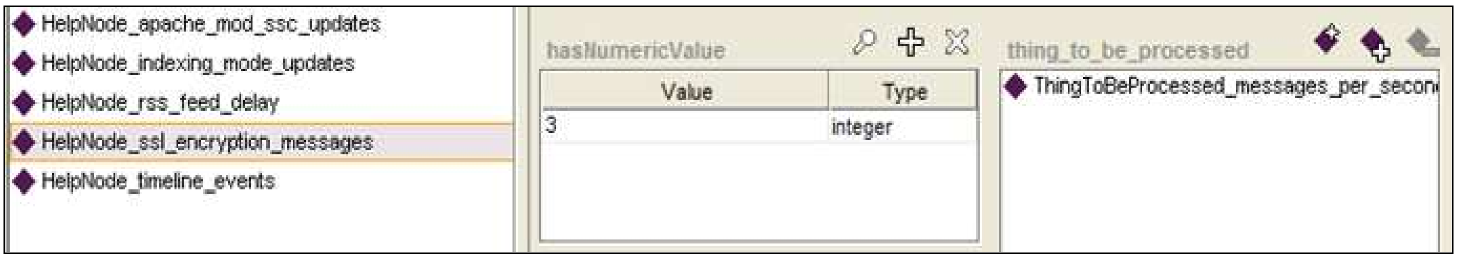


Figure 33: Modeling numeric statements about things to be processed

The following screenshots show the definition of the user-created help node and how it is being related to the individual "SSL encryption" in Protegé:



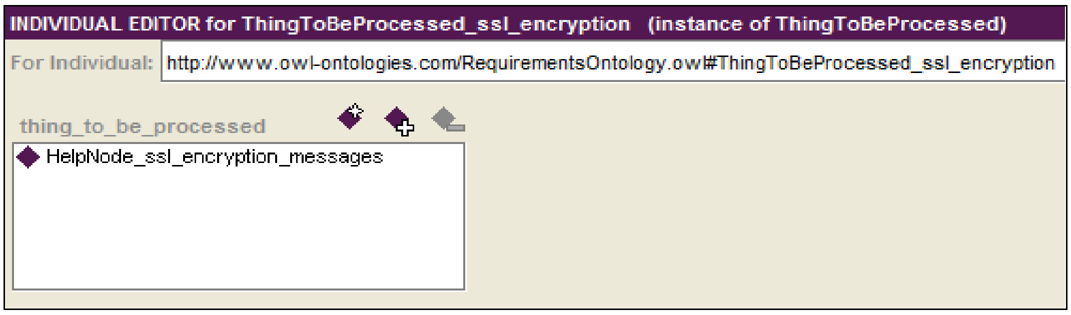
HelpNode_apache_mod_ssc_updates
HelpNode_indexing_mode_updates
HelpNode_rss_feed_delay
HelpNode_ssl_encryption_messages
HelpNode_timeline_events

hasNumericValue

Value	Type
3	integer

thing_to_be_processed
ThingToBeProcessed_messages_per_secon

Figure 34: Creating a help node in Protegé



INDIVIDUAL EDITOR for ThingToBeProcessed_ssl_encryption (instance of ThingToBeProcessed)

For Individual: http://www.owl-ontologies.com/RequirementsOntology.owl#ThingToBeProcessed_ssl_encryption

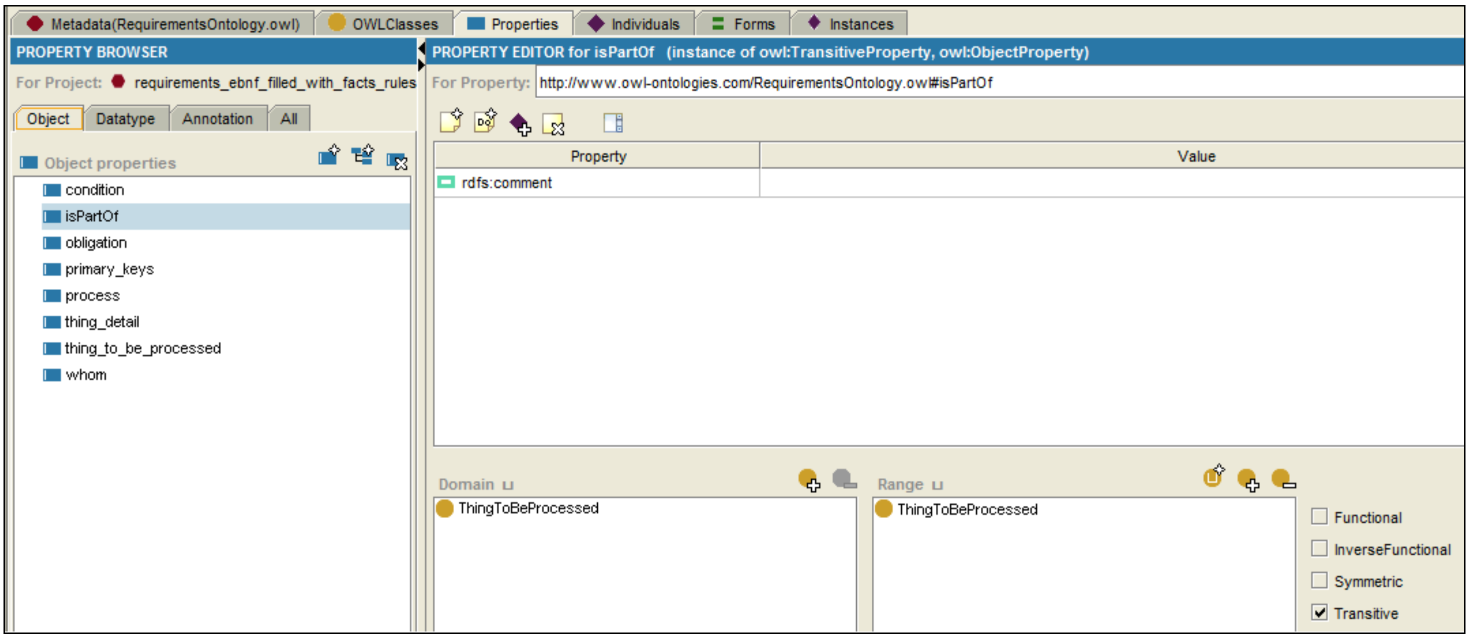
thing_to_be_processed
HelpNode_ssl_encryption_messages

Figure 35: Connecting "SSL encryption" with the help node

- Another important possibility in ontology modeling is to manually define relationships between elements.

  For example the user could create a property called “isPartOf” to describe which elements are a part of each other.

  The following screenshots show the definition of the (transitive) property “isPartOf”, and the statement that “the sprint definition panel is part of the administration panel”:



Metadata(RequirementsOntology.owl)
OWLClasses
Properties
Individuals
Forms
Instances
PROPERTY BROWSER
For Project: requirements_ebnf_filled_with_facts_rules
Object
Datatype
Annotation
All
Object properties
PROPERTY EDITOR for isPartOf (instance of owl:TransitiveProperty, owl:ObjectProperty)
Property
Value
Domain
Range

For Property: http://www.owl-ontologies.com/RequirementsOntology.owl#isPartOf

- condition
- isPartOf
- obligation
- primary_keys
- process
- thing_detail
- thing_to_be_processed
- whom

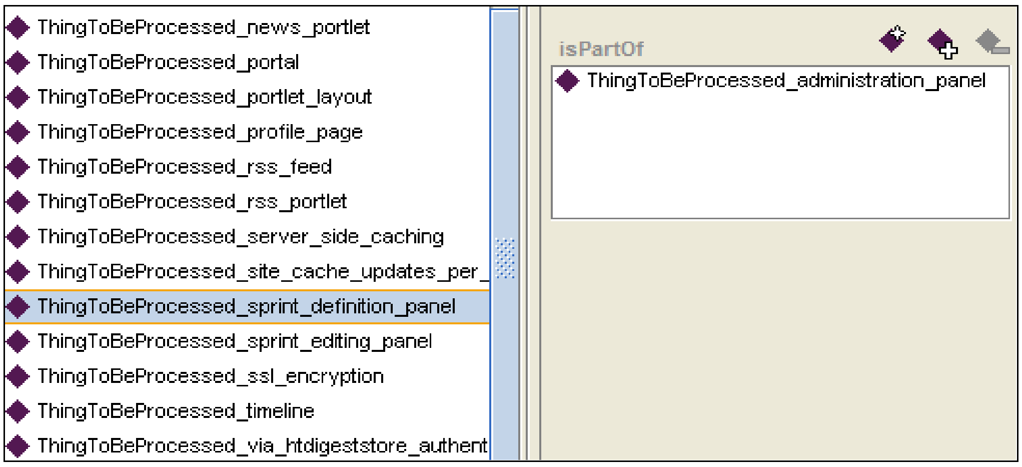
Property	Value
rdfs:comment	

Domain: ThingToBeProcessed

Range: ThingToBeProcessed

☐ Functional
☐ InverseFunctional
☐ Symmetric
☑ Transitive

Figure 36: Creating an object property in Protegé



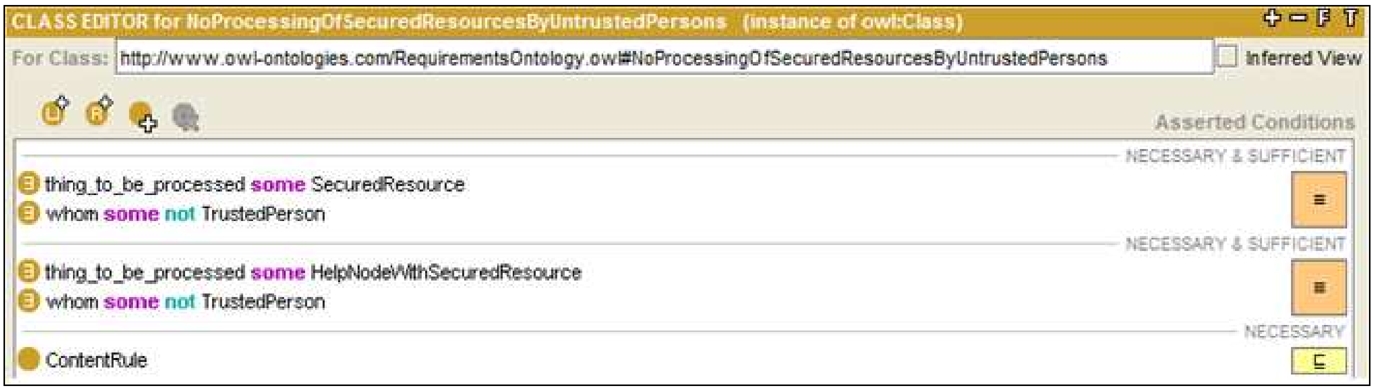
- ThingToBeProcessed_news_portlet
- ThingToBeProcessed_portal
- ThingToBeProcessed_portlet_layout
- ThingToBeProcessed_profile_page
- ThingToBeProcessed_rss_feed
- ThingToBeProcessed_rss_portlet
- ThingToBeProcessed_server_side_caching
- ThingToBeProcessed_site_cache_updates_per_
- ThingToBeProcessed_sprint_definition_panel
- ThingToBeProcessed_sprint_editing_panel
- ThingToBeProcessed_ssl_encryption
- ThingToBeProcessed_timeline
- ThingToBeProcessed_via_htdigeststore_authent

isPartOf: ThingToBeProcessed_administration_panel

Figure 37: Relating two individuals with the "isPartOf" property

3. As next step the user has to model all necessary business constraints (or "rules") and documentation guidelines for conflict types "CRC" and "CRG" as ontology classes. If these constraints have been modeled correctly, the reasoner is able to automatically compute the requirements that conflict with them.
   They must be modeled by using the "necessary & sufficient" conditions in the class definitions in "Protegé". There are countless possibilities to do that, so I will only bring some examples here:

- As an example a business constraint could constrict that some resources shall not be accessible by anybody, by stating that "No requirement shall allow secured resources to be processed by untrusted persons."
  Assuming that the user has already defined the "secured resources" and the "trusted persons" as glossary terms in the ontology (like shown in the previous step), the restrictions in the class definition can look like shown in the following screenshot:


CLASS EDITOR for NoProcessingOfSecuredResourcesByUntrustedPersons (instance of owl:Class)

For Class: http://www.owl-ontologies.com/RequirementsOntology.owl#NoProcessingOfSecuredResourcesByUntrustedPersons

Inferred View

Asserted Conditions

NECESSARY & SUFFICIENT

thing_to_be_processed some SecuredResource
whom some not TrustedPerson

NECESSARY & SUFFICIENT

thing_to_be_processed some HelpNodeWithSecuredResource
whom some not TrustedPerson

NECESSARY

ContentRule

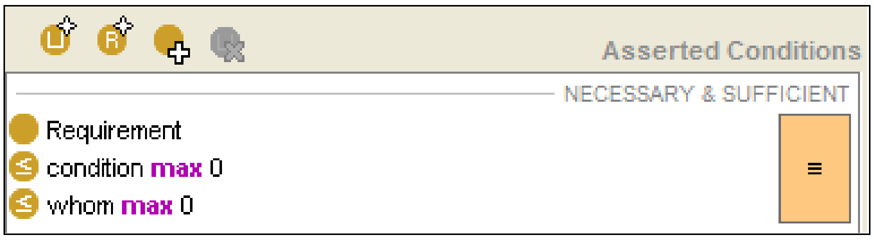
Figure 38: Defining a business contraint in Protegé

The "necessary & sufficient" definitions state that "every requirement, that is in a 'thing_to_be_processed' relationship with a 'SecuredResource', and is at the same time in a 'whom' relationship with an individual that is NOT a 'TrustedPerson', shall be computed by the reasoner as an instance of this class".
Because the "thing_to_be_processed" relation can be modeled directly or indirectly over a help node, two "necessary & sufficient" definitions are necessary here.

- It is not only possible to constrain the contents of the requirements, but also their formal structure through the usage of “documentation guidelines”. Their creation works pretty similar to the modeling of business constraints.

  As an example, a guideline could state that “all requirements must define either a ‘condition’ or a ‘person’ (or both)”. This guideline guarantees a minimum of preciseness in the requirements.

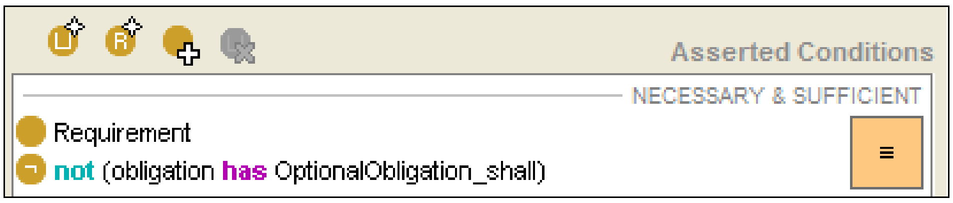
  To model this guideline in the ontology, it is necessary to define a class with a “necessary & sufficient” definition that includes all requirements that have maximal zero “condition” and maximal zero “whom” relationships. An example is shown in the following screenshot:



Requirement
condition **max** 0
whom **max** 0

Figure 39: Modeling a documentation guideline in Protegé

- Another example documentation guideline is: “All requirements must use the obligation word ‘shall’”. An ontology class that models this rule defines a “necessary & sufficient” condition that includes all requirements that do not contain the obligation type “shall”, like shown in the following screenshot:



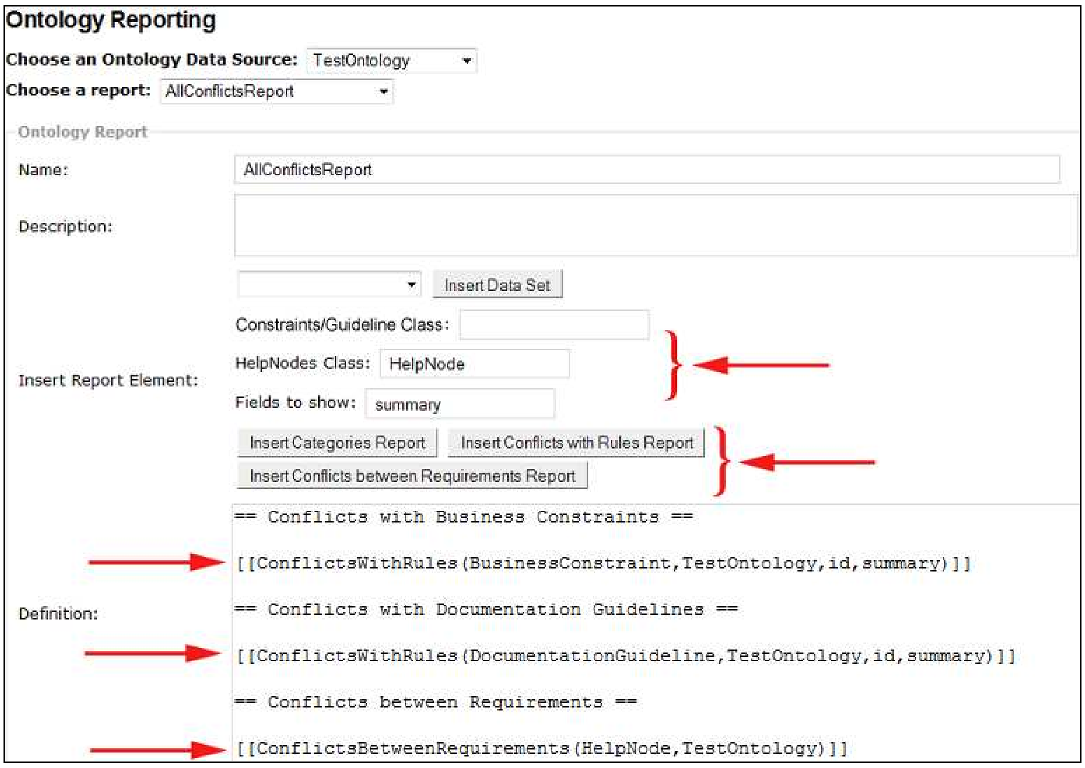
Requirement
not (obligation **has** OptionalObligation_shall)

Figure 40: Documentation guideline that checks if all requirements use the obligation word "shall"

Performing the conflict analysis

1. When everything is prepared, it is necessary to deliver the data to the ReportGenerator. To do so the user first has to specify his ontology as a data source with type “ontology” and a self-given name in the menu “data sources”, if not already done (similar to section 3.2.2).

2. Now he can create a report with the OntologyReporting plug-in by inserting placeholder elements that will automatically be filled with the “conflicts between requirements” and the “conflicts between requirements and rules” listings.
   To do so, he first has to fill in the correct values for the names of the classes and click the corresponding buttons. In the following screenshot, the first two red arrows mark the fields that shall contain the class names. The buttons that create the placeholders for the automatically generated listings are marked by the last three arrows:



Ontology Reporting

Choose an Ontology Data Source: TestOntology

Choose a report: AllConflictsReport

Ontology Report

Name: AllConflictsReport

Description:

Insert Report Element:

Constraints/Guideline Class:

HelpNodes Class: HelpNode

Fields to show: summary

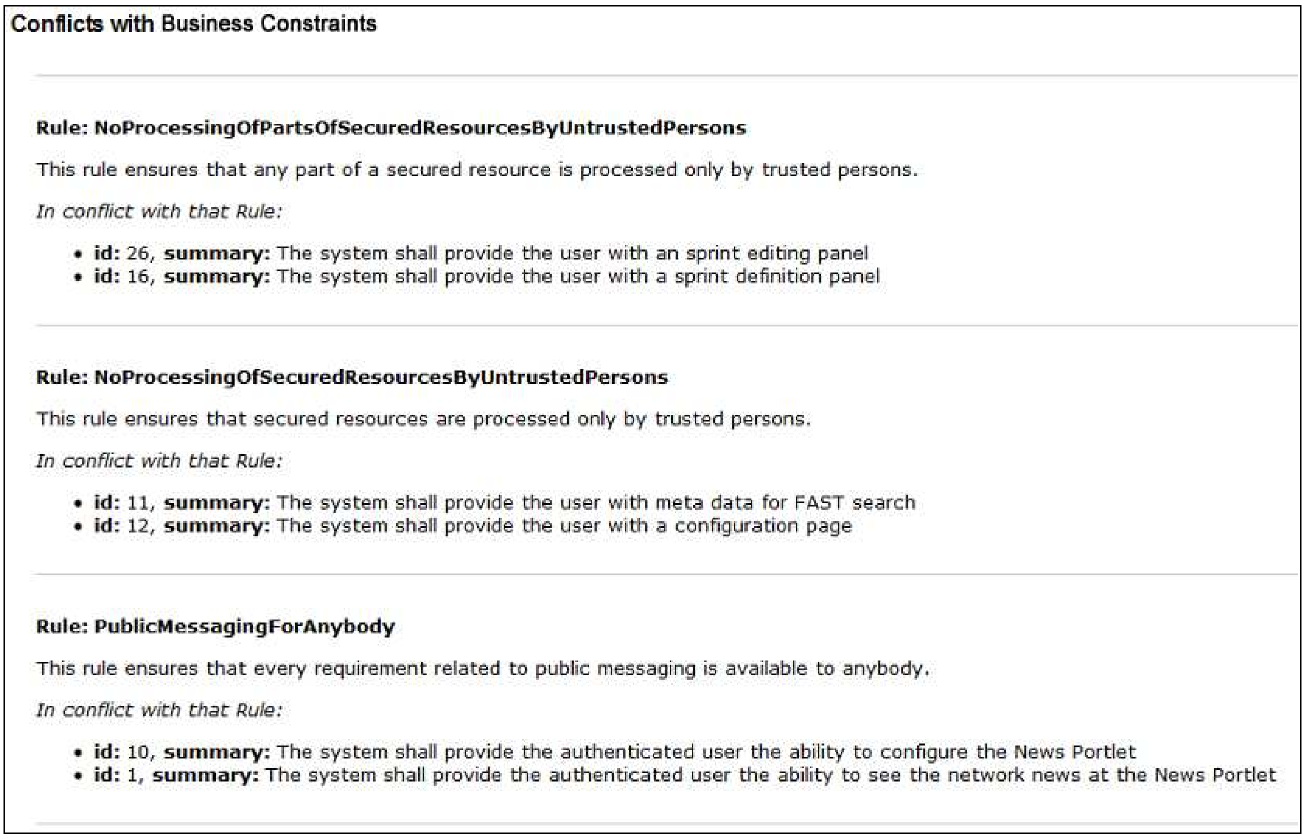
Definition:

```
== Conflicts with Business Constraints ==
[[ConflictsWithRules(BusinessConstraint,TestOntology,id,summary)]]
== Conflicts with Documentation Guidelines ==
[[ConflictsWithRules(DocumentationGuideline,TestOntology,id,summary)]]
== Conflicts between Requirements ==
[[ConflictsBetweenRequirements(HelpNode,TestOntology)]]
```

Insert Data Set
Insert Categories Report
Insert Conflicts with Rules Report
Insert Conflicts between Requirements Report

Figure 41: Creating a report template for the automated conflict analysis

3. If the user has finished editing his report template, he can now see a preview of the generated report, or download it in different formats (like described in section 3.2.2). The following screenshots show examples of generated conflict reports:



Conflicts with Business Constraints

Rule: NoProcessingOfPartsOfSecuredResourcesByUntrustedPersons

This rule ensures that any part of a secured resource is processed only by trusted persons.

In conflict with that Rule:

- **id:** 26, **summary:** The system shall provide the user with an sprint editing panel
- **id:** 16, **summary:** The system shall provide the user with a sprint definition panel

Rule: NoProcessingOfSecuredResourcesByUntrustedPersons

This rule ensures that secured resources are processed only by trusted persons.

In conflict with that Rule:

- **id:** 11, **summary:** The system shall provide the user with meta data for FAST search
- **id:** 12, **summary:** The system shall provide the user with a configuration page

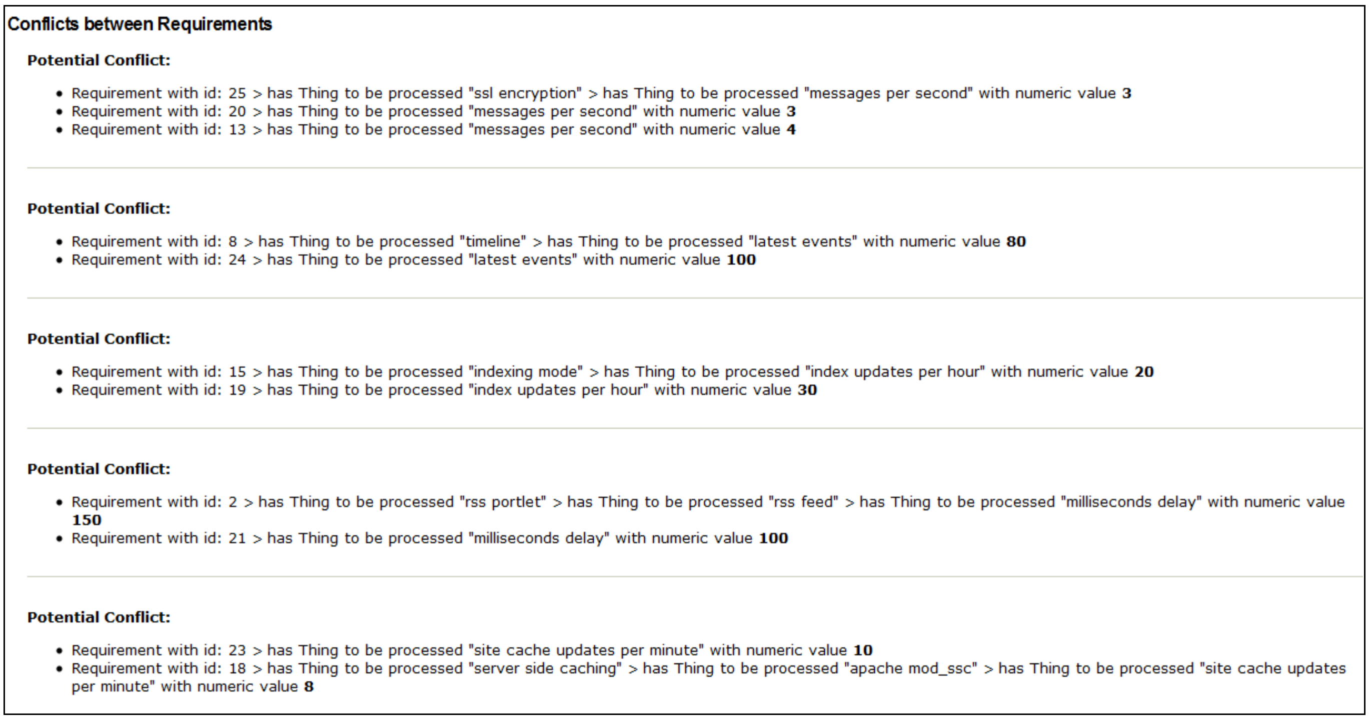
Rule: PublicMessagingForAnybody

This rule ensures that every requirement related to public messaging is available to anybody.

In conflict with that Rule:

- **id:** 10, **summary:** The system shall provide the authenticated user the ability to configure the News Portlet
- **id:** 1, **summary:** The system shall provide the authenticated user the ability to see the network news at the News Portlet

Figure 42: Generated example report showing conflicts with business constraints


Conflicts between Requirements

Potential Conflict:

- Requirement with id: 25 > has Thing to be processed "ssl encryption" > has Thing to be processed "messages per second" with numeric value **3**
- Requirement with id: 20 > has Thing to be processed "messages per second" with numeric value **3**
- Requirement with id: 13 > has Thing to be processed "messages per second" with numeric value **4**

Potential Conflict:

- Requirement with id: 8 > has Thing to be processed "timeline" > has Thing to be processed "latest events" with numeric value **80**
- Requirement with id: 24 > has Thing to be processed "latest events" with numeric value **100**

Potential Conflict:

- Requirement with id: 15 > has Thing to be processed "indexing mode" > has Thing to be processed "index updates per hour" with numeric value **20**
- Requirement with id: 19 > has Thing to be processed "index updates per hour" with numeric value **30**

Potential Conflict:

- Requirement with id: 2 > has Thing to be processed "rss portlet" > has Thing to be processed "rss feed" > has Thing to be processed "milliseconds delay" with numeric value **150**
- Requirement with id: 21 > has Thing to be processed "milliseconds delay" with numeric value **100**

Potential Conflict:

- Requirement with id: 23 > has Thing to be processed "site cache updates per minute" with numeric value **10**
- Requirement with id: 18 > has Thing to be processed "server side caching" > has Thing to be processed "apache mod_ssc" > has Thing to be processed "site cache updates per minute" with numeric value **8**

Figure 43: Generated example report showing numerical conflicts between requirements

3.3.4 Technical Description: Requirements conflict analysis

I will now describe the internal view and state how the conflict check is performed by OntRep. This part only describes the steps that are executed automatically and does not go into details regarding the manual steps from section 3.3.3 that need to be performed by the user:

Fetching process

1. The filling of the requirements into the ontology works pretty similar to the process described in section 3.2.3. But additionally, when performing the conflict check, there must exist an attribute in all records that describes the text of the requirements using the specified EBNF grammar. If existing, the OntologyFetcher loads the value of this attribute after it saved the requirement in the category.
2. This value in EBNF grammar is being analyzed by a parsing module. If it uses valid grammar, the parser splits up the text into the different parts (e.g. “condition”,

"process_verb", "thing_to_be_processed",...) and returns them to the OntologyFetcher.
3. The OntologyFetcher now creates an individual for each of the returned parts and saves them in the matching subclass of the ontology class "EBNF_TemplateElement" (like shown in Figure 30).
4. Now the user has to model all relevant facts and constraints in the ontology like described in section 3.3.3.

Check for conflicts of type "CRC" and "CRG"

The check for conflicts between requirements and business constraints or documentation guidelines starts when the user generates the corresponding report. The technical details behind this check are not complicated, because the main work for this check is being done by the ontology reasoner:

1. The OntologyReporting plug-in loads the ontology that shall be checked for conflicts.
2. The inferred model of the ontology is being computed by using the reasoner "Pellet". The inferred model includes all statements that can be computed through ontology reasoning.
   In this process, all requirements that conflict with business constraints or documentation guidelines are automatically being related with the ontology classes that represent those constraints or guidelines (if modeled correctly).
3. The OntologyReporting plug-in iterates through all ontology classes that represent business constraints or documentation guidelines, and loads the requirements they contain after the reasoning process.
4. It generates a HTML page that shows all constraints and guidelines together with their descriptions (if existing as "rdfs:comment") and the requirements that are in conflict with them, and delivers this page to the user.

Check for conflicts of type “CRR”

Preceding note:

During fetching the OntologyFetcher has saved all requirements that contain numerical values by using anonymous help nodes. It related the requirement with a help node, which has been related itself with the actual “thing to be processed” and the corresponding numeric value. This way, the same “thing to be processed” can be related to different requirements with different numeric values. The following diagram shows the coherences:

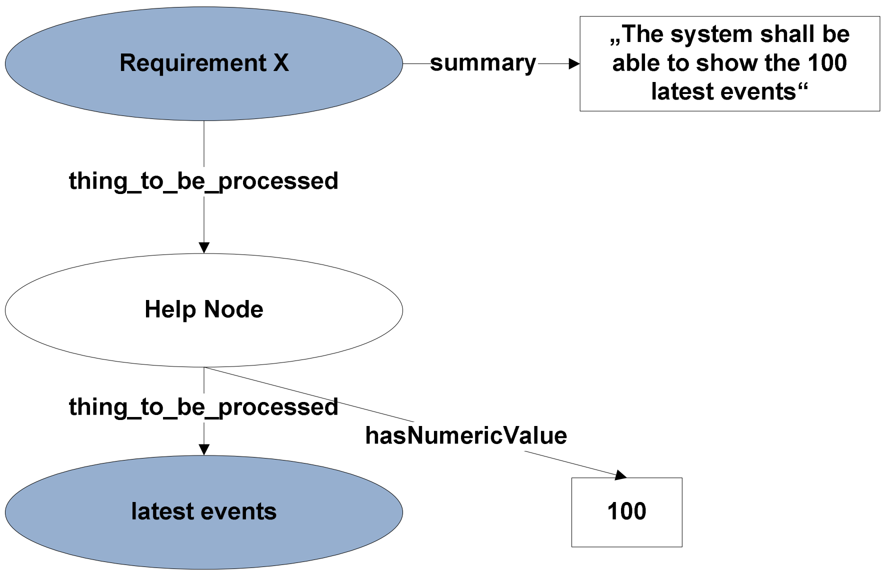


Figure 44: Saving requirements that make numeric statements about things in the ontology with help nodes

1. When the user wants to generate a “CRR” conflicts report, the OntologyReporting plug-in iterates through all help nodes. It loads all individuals that are in the “HelpNode” ontology class.
2. The OntologyReporting plug-in now checks for different help nodes that relate the same “thing to be processed” with different numeric values.
3. Now it searches backwards from those help nodes for the requirements that are related with them. The requirements can possibly be connected over several nodes with multiple “thing to be processed” relationships.
4. Finally, the OntologyReporting plug-in produces HTML that lists the requirements that are attaching different numerical values to the same “thing to be processed” in

several groups of conflicts. It also shows the paths of nodes that lead from the requirements to the related “things to be processed”. Then this HTML page is returned to the user.

3.4 Usage Scenario 3: General reporting tasks

The third usage scenario of OntRep deals with its standard reporting capabilities, without any semantic interpretation of data. The general reporting functionalities provided by the ReportGenerator work pretty straightforward and are based on the separation of data sources, queries (also called “datasets”) and report templates.
A data source can be accessed through different queries, and a query can be embedded in any number of report templates. This separation allows a high rate of reusability of already existing resources. To design a report template, the user can utilize Trac’s wiki formatting syntax and all of its elements.
The following section lists the goals for general reporting that shall be attained by OntRep in more detail. But the focus of this master thesis is on the semantic aspects of OntRep, so I won’t go into any more details here regarding its general reporting capabilities.

3.4.1 Goals for general reporting tasks

- *OntRep shall be able to access any kind of data source there is a driver for. The drivers shall be extendable.*
  The prototype shall be equipped with an architecture that allows the access to data sources through different drivers, e.g. a driver for “MySQL” databases, another driver for XML data,… etc. This architecture shall also allow to add new drivers for new data sources. Drivers for at least one common data source shall be delivered with the prototype (like the driver for a common database).
- *The user shall be able to define custom queries for data sources.*
  A custom query for example could be a SQL statement.
- *The user shall be able to define joined queries from existing queries.*
  There shall be a possibility that allows the user to construct joins from the results of different queries. These joins must be processed directly from the prototype (and not from the data source).

This shall offer the possibility to integrate and compare the data from different data sources. With this feature a user could for example directly compare the average salaries from an online database with the salaries from a local database.

The user shall be provided with the ability to compose left inner, left outer and full outer joins.

- *The user shall be able to flexibly define the contents of each report template.*

  The user shall have the possibility to define each report by specifying a report template. The contents of this template shall be definable in a flexible way, using static content (like text, headers, images) and dynamic content (like queries that will be filled with the query results when generating the report).

- *The user shall be able to include existing report templates into another report template.*

  This modularity offers the user the ability to reuse existing report templates and compose new reports from them.

- *The user shall be provided with the possibility to choose the visual formats for queries. These visual formats shall be extendable.*

  There are different options of how to visually render the results of a query. Examples are: As a list, in a table or as a chart. The user shall be able to choose from different formats, and there shall be the possibility to add new formats to the prototype. The prototype shall be delivered with the most common formats for result presentation, like table and list.

- *The user shall be able to preview the generated report result.*

  This feature allows to make adjustments while developing the report template.

- *The user shall be able to choose from different formats for the generated report. These formats shall be extendable.*

  When the report template is finished and the user wants to generate the report, there shall be different formats he can choose from, like for example PDF, HTML, DOC,…

  The prototype does not have to support all these formats innately, but shall offer the possibility to extend the available formats.

3.5 Overview of technologies used in OntRep

OntRep has been implemented by combining several different technologies. I will now give an overview of the most important technologies that have been used and their versions:

- **Python 2.4.4**
  The dynamic programming language Python is the core fundament for OntRep. Trac is implemented in Python, and so all plug-ins for Trac must also be based on Python [Python].
- **Trac 0.11.4**
  Trac is a web-based platform for configuration and project management (also see section 3.1.1 for more details). It is the environment in which the OntRep plug-ins are embedded.
- **Genshi 0.5.1**
  Genshi is a library for Python that is being used to generate the HTML interfaces the user interacts with, and handle the HTTP communication between the plug-ins and the user [Genshi].
- **Jena 2.5.7**
  Jena is a collection of libraries for the programming language Java. It provides a broad range of functionalities to read and modify ontologies of different languages [Jena].
- **JPype 0.5.3**
  Jena is the most comprehensive framework for processing ontologies that is available for free. But it only exists in Java, though OntRep had to be implemented using Python. So I decided to use JPype, which allows Python programs access to Java libraries [JPype].
- **WordNet**
  WordNet is the lexical database that is being used to find synonyms and hyponyms during the automated categorization process (also see section 2.4 for more details) [WordNet, 1995].

- **pywordnet 2.0.1**
  pywordnet is a Python library that is being used to easily access the WordNet database from Python [pywordnet].
- **Protegé 3.4 with OWL plug-in**
  The Protegé editor is an open source application that can be used to analyze and edit ontologies. A plug-in for Protegé allows to edit OWL ontologies. Its usage has been described partially in sections 3.2.2 and 3.3.3, but please see the website for more information on Protegé [Protegé].
- **Pellet 1.5.2**
  Pellet is an open source reasoner for OWL [Pellet]. It is being accessed through functions from the Jena libraries (which are being accessed through JPype themselves). Please see section 1.3.2 for more details on reasoning.

3.6 Research questions

OntRep shall help me to answer the following research questions:

RQ1:
Which benefits does an ontology-based reporting approach bring regarding the categorization and conflict analysis of software requirements?

RQ2:
What efforts have to be realized in order to prepare ontology-based reporting?

Based on my personal experiences with automated tool support, I assumed that OntRep could help to increase the efficiency and quality of conflict analysis, and the effectiveness for requirements categorization. Due to lacks in natural language processing techniques I was not sure if OntRep could help to increase the quality of requirements categorization.
But I also assumed that OntRep reduces the effort for both, requirements categorization and conflict analysis.
However, as the introduction of a new approach also brings new complexities, its effects on a software engineering process needs to be investigated empirically in a relevant setting.

In order to address the research questions, I derived the following **independent variables** (according to [ViSEK, 2002]) to consider for evaluation:

- The number of given requirements influences the necessary effort for their categorization and check for conflicts.
- The number of categories used to categorize the requirements.
- The total number of true requirement conflicts (of exactly specified conflict types) existing in the given set of requirements. A perfect approach would find 100% of the true requirement conflicts.
- The chosen approach for requirements categorization and conflict analysis: Primarily I distinguish between the automated and manual approach. But also the exact details of each approach and how it is performed has effects on the results of the evaluation.

Dependent variables, that result from the evaluation and need to be measured, are:

- Number of identified conflicts:
  This number consists of two measurements: Number of correctly identified conflicts (for measuring the effectiveness of an approach) and false positives (number of conflicts that have been identified but are not true conflicts).
- True conflicts that have not been identified (false negatives):
  This number can be calculated by subtracting the number of correctly identified conflicts from the total number of true requirement conflicts.
- Correctness of requirements categorization:
  Regarding the categorization task two kinds of errors can occur: (1) Requirements have been assigned to a wrong category, and (2) requirements have not been assigned to a category they actually belong to. In order to measure these parameters, I took the manual categorization results of an requirements engineering expert as a reference.

Beside these parameters, I also measured the efforts that had to be invested for requirements categorization and conflict analysis. This includes preparation effort (e.g. modeling the ontology that is used for categorization and conflict analysis), categorization effort, and conflict analysis effort. The performed case study is described in more details in the following section.

4 Evaluation

This section describes the methodology of the evaluation that has been performed to analyze the costs and benefits of the approach “OntRep”.

4.1 Introduction & goals of evaluation

The primary goal of the evaluation was to answer the research questions presented in section 3.6, and was performed as a case study at Siemens Austria.
The evaluation consisted of two parts. Each of these parts is responsible for evaluating a different aspect of OntRep. The two different parts of the evaluation are related to each other, but are not mutually dependent. This means that the results from one part can be analyzed without the results from the other. So these parts can be viewed as two autonomous evaluations.

The two parts of the evaluation were:

- Evaluate the automated categorization of requirements in comparison to a manual approach.
- Evaluate the automated conflict check for requirements in comparison to a manual approach.

The purpose of both parts of the evaluation was to compare the ontology-based approach with a manual performance of the tasks in respect to efforts and result quality. The test data for the evaluation was based on the Siemens project “TechnoWeb2.0”.

4.2 Description of evaluation project and test data: “TechnoWeb2.0”

The “TechnoWeb2.0” project is a software development project at Siemens Austria. Its goal is to design and implement a web application that serves as a platform for communication and networking between different technology experts inside the Siemens company. It uses the Java technology “Liferay”, where so-called “Portlets” make up the components of the web

application. The project is being realized in an agile way using the software development process “SCRUM”, and the configuration & project management platform Trac [Trac]. In Trac, all requirements, tasks and bugs can be saved and organized as so-called “Tickets”.
This type of project suited well to evaluate OntRep, because OntRep has been implemented as a Trac plug-in and was easy to integrate with the project’s tool infrastructure.

For the purpose of the evaluation the tickets of the project have been analyzed and a subset of them has been transferred into the requirements management tool “RequisitePro”, to be able to generate a clean software requirements specification (SRS) document. Furthermore, I added some self-created requirements to this document to be able to evaluate certain combinations and provoke conflicts between requirements.
Some statistics about the SRS document:

- Number of requirements: 23
- Number of technical constraints: 7
- Number of business constraints: 4
- Number of glossary term definitions: 7
- Number of documentation guidelines: 4

4.3 Case study design

The following diagram shows the whole workflow that had to be performed to prepare and execute the case study:

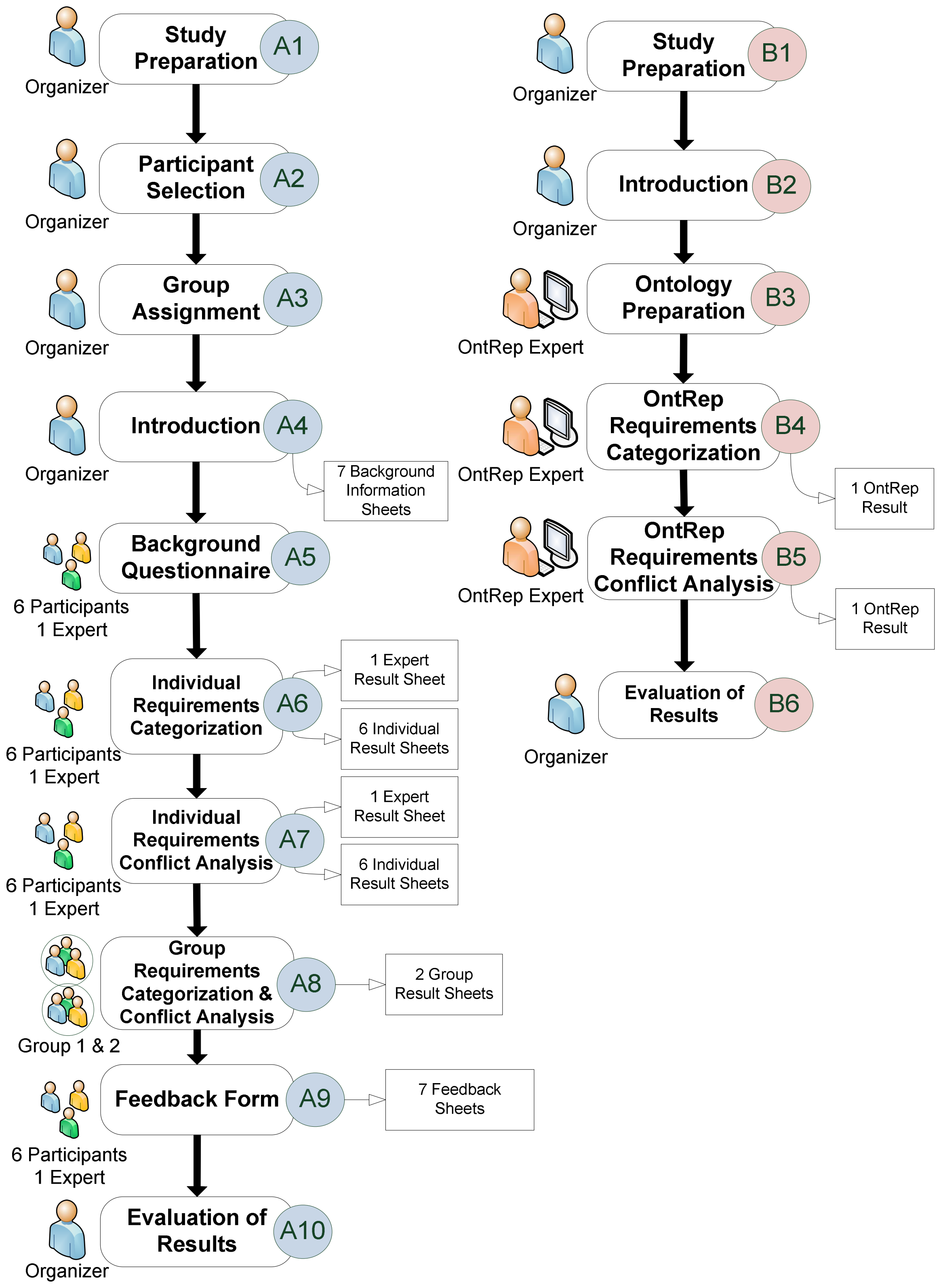


Figure 45: Workflow of the case study to evaluate OntRep

The diagram shows the two workflows that have been performed to evaluate and compare (1) the manual and (2) the ontology-based execution of requirements categorization and conflict analysis. The two workflows had the same requirements data as input, but were independent from each other apart from that. I will now describe each step of the workflow in more detail:

A1 & B1: Study Preparation

At first it was necessary to plan the evaluation and prepare all material necessary for its execution. This included the following steps to take:

- Create an accurate evaluation plan.
- Prepare the requirements and the conflicts as test data (based on the "TechnoWeb2.0" project).
- Create a questionnaire for background information of the participants.
- Create the forms for manual execution of requirements categorization and conflict check.
- Create a feedback form.
- Deploy OntRep software on a computer.

A2 & A3: Participant selection and group assignment:

Next I had to choose which employees at Siemens shall participate in the evaluation. I found 6 participants who had similar experience in software development and project management, and were willing to participate. They had basic know-how in requirements engineering. Furthermore, they were randomly assigned into two different groups, which was necessary for step A8. In addition, one expert at requirements engineering participated in the evaluation (on his own and not in any group).

The part of the OntRep tool expert has been performed by myself.

A4 & B2: Introduction

The "Introduction" step has been inserted to have a clear separation between the preparations and the start of the evaluation. In this phase, the participants gathered at Siemens, have been welcomed and briefed about the tasks and how to perform them. The time measurement did not start until all participants had gained a clear understanding of the tasks.

The introduction step for the ontology-based prototype began when all necessary study preparations have been met. I set everything up to be able to start time measurements, before the practical part to evaluate OntRep started.

A5: Background questionnaire

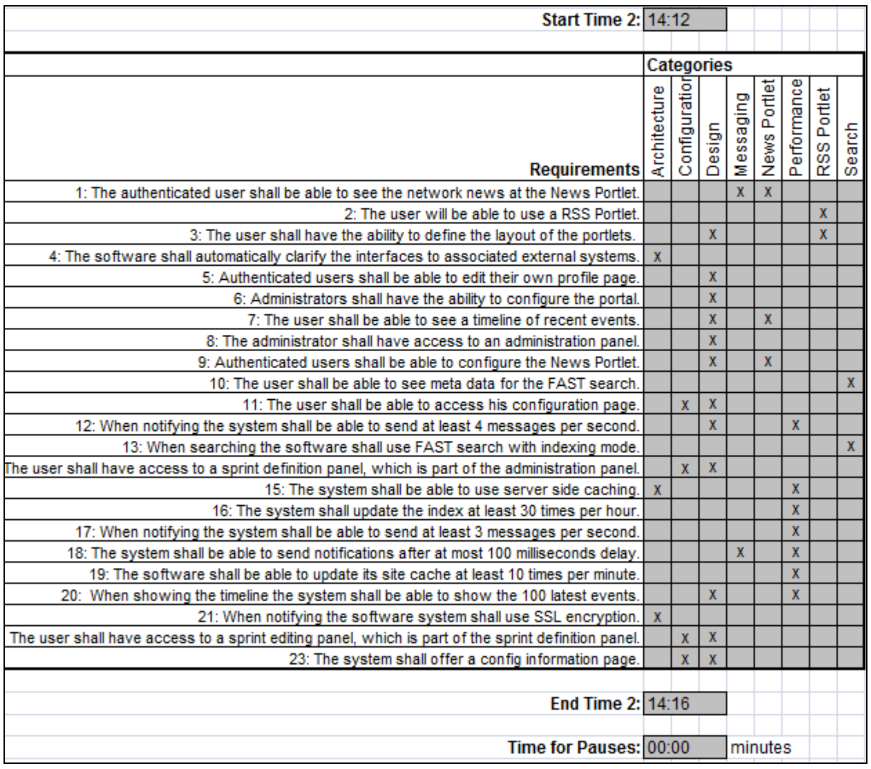
Each participant had to fill in personal information before starting to perform the actual tasks of the evaluation. This questionnaire included the following questions:

- What are your main roles in software development projects?
- How do you rate your ability to understand English documents? (low – some – advanced – expert)
- How many years of experience in project management do you have? (less than 1 year – between 1 and 3 years – between 3 and 5 years – more than 5 years)
- What is your level of requirements engineering know-how? (low – some – advanced – expert)
- What is your level of software development know-how? (low – some – advanced – expert)

A6: Individual requirements categorization

Now the actual tasks for the manual part of the evaluation started. It was performed by using an MS Excel sheet. This sheet included detailed instructions (additionally to the instructions given by the evaluation organizer) on the tasks, and exactly marked the fields the participants had to fill in.

It included fields for start, end time and paused time, a description of the different categories to choose from and a table to relate the requirements with the categories. The following screenshot shows the main part of the form:



Start Time 2: 14:12

	Categories							
Requirements	Architecture	Configuration	Design	Messaging	News Portlet	Performance	RSS Portlet	Search
1: The authenticated user shall be able to see the network news at the News Portlet.				X	X			
2: The user will be able to use a RSS Portlet.							X	
3: The user shall have the ability to define the layout of the portlets.			X				X	
4: The software shall automatically clarify the interfaces to associated external systems.	X							
5: Authenticated users shall be able to edit their own profile page.			X					
6: Administrators shall have the ability to configure the portal.			X					
7: The user shall be able to see a timeline of recent events.			X		X			
8: The administrator shall have access to an administration panel.			X					
9: Authenticated users shall be able to configure the News Portlet.			X		X			
10: The user shall be able to see meta data for the FAST search.								X
11: The user shall be able to access his configuration page.		X	X					
12: When notifying the system shall be able to send at least 4 messages per second.			X			X		
13: When searching the software shall use FAST search with indexing mode.								X
The user shall have access to a sprint definition panel, which is part of the administration panel.		X	X					
15: The system shall be able to use server side caching.	X					X		
16: The system shall update the index at least 30 times per hour.						X		
17: When notifying the system shall be able to send at least 3 messages per second.						X		
18: The system shall be able to send notifications after at most 100 milliseconds delay.				X		X		
19: The software shall be able to update its site cache at least 10 times per minute.						X		
20: When showing the timeline the system shall be able to show the 100 latest events.			X			X		
21: When notifying the software system shall use SSL encryption.	X							
The user shall have access to a sprint editing panel, which is part of the sprint definition panel.		X	X					
23: The system shall offer a config information page.		X	X					

End Time 2: 14:16

Time for Pauses: 00:00 minutes

Figure 46: The form for manual requirements categorization

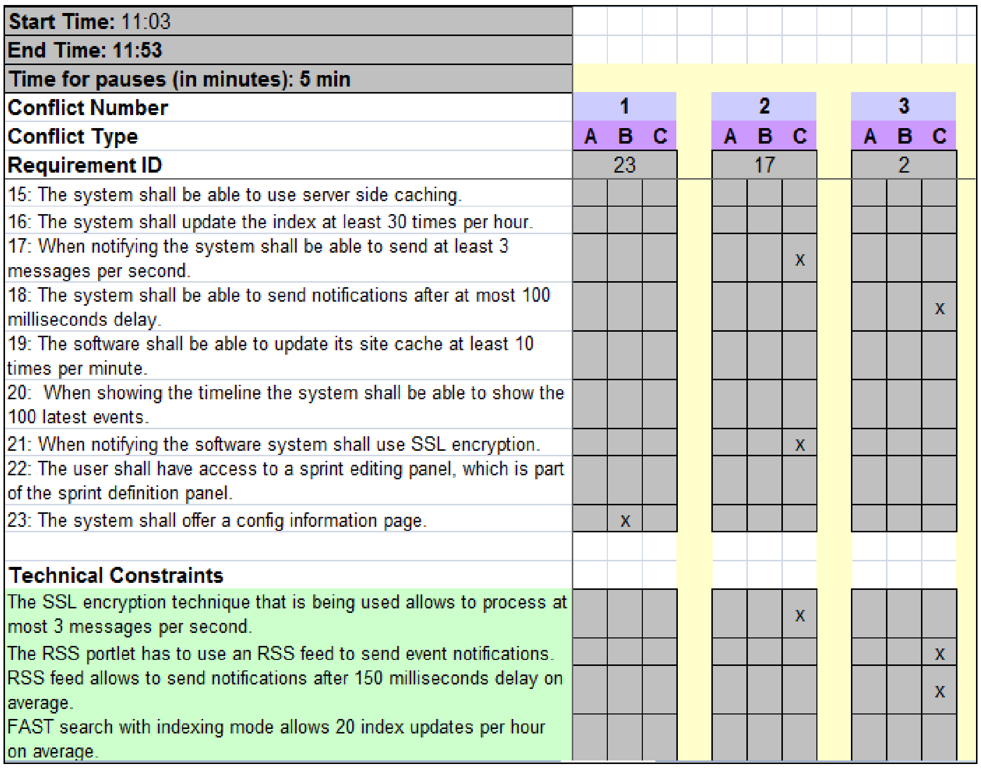
The form has been filled by each participant individually, and additionally by the requirements engineering expert. So the results were 6 sheets from the participants and 1 sheet from the expert.

A7: Individual requirements conflict analysis

The next step was to perform the manual conflict analysis. Therefore the participants were introduced into the task by the organizer, and additionally read the detailed instructions with examples on how to fill out the form contained in the MS Excel sheet.

The form exactly marked which fields the participants had to fill in. It also contained fields for start, end and paused time. The form structured the different elements (like requirements, business constraints, glossary terms,…) in an understandable way to be able to minimize formal errors.

The participants just had to mark the combination of requirements and all other elements that they believed to cause a conflict in the column of the corresponding conflict type with an "X". The following screenshot shows parts of the form:



Start Time: 11:03									
End Time: 11:53									
Time for pauses (in minutes): 5 min									
Conflict Number		1			2			3	
Conflict Type	A	B	C	A	B	C	A	B	C
Requirement ID		23			17			2	
15: The system shall be able to use server side caching.									
16: The system shall update the index at least 30 times per hour.									
17: When notifying the system shall be able to send at least 3 messages per second.						x			
18: The system shall be able to send notifications after at most 100 milliseconds delay.									x
19: The software shall be able to update its site cache at least 10 times per minute.									
20: When showing the timeline the system shall be able to show the 100 latest events.									
21: When notifying the software system shall use SSL encryption.						x			
22: The user shall have access to a sprint editing panel, which is part of the sprint definition panel.									
23: The system shall offer a config information page.		x							
Technical Constraints									
The SSL encryption technique that is being used allows to process at most 3 messages per second.						x			
The RSS portlet has to use an RSS feed to send event notifications.									x
RSS feed allows to send notifications after 150 milliseconds delay on average.									x
FAST search with indexing mode allows 20 index updates per hour on average.									

Figure 47: The form for manual conflict analysis

Again, the form has been filled by each participant individually, and additionally by the requirements engineering expert. So the results were 6 sheets from the participants and 1 sheet from the expert.

A8: Group requirements categorization and conflict analysis

Now the 6 participants grouped together into two different teams (that had been defined in step A3) and were assigned the task to harmonize their individual results and produce a combined group result. This procedure should simulate a review meeting, like it is very common in software development projects.

The two groups worked separately. The results of this step were two different group result sheets.

A9: Feedback form

Finally the participants were asked to fill in a form with personal feedback regarding the evaluation: Where did they have problems? How easy was it to perform the tasks?

I made two separate fields to distinct the feedback for requirements categorization and conflict analysis.

B3: Ontology preparation

As next step in the ontology-based evaluation process, the tool expert (myself) now started with time measuring and prepared the ontology according to the needs of the task. This included:

- Converting the given requirements into the specified EBNF grammar.
- Preparing the ontology classes that serve as categories.
- Defining synonyms and negated synonyms for those ontology classes.

B4: OntRep requirements categorization

In this step the tool expert loaded the converted requirements into the tool, set all necessary parameters and initiated the automated categorization process.

The threshold parameters were set to the value 1. This was the optimal value in this case, because the texts of the requirements were quite short, and so there were almost no repetition of words or synonymic words (which is necessary to reach thresholds higher than 1).

The time necessary to perform the fetching process and report generation was also measured.

B5: OntRep requirements conflict analysis

Now the automated categorization process had been finished, the evaluation of the automated conflict check started. Therefore I performed the steps described in section 3.3.3. This included:

- The modeling of glossary terms in the ontology.
- The modeling of other relevant facts in the ontology (like technical constraints).
- The modeling of business constraints in the ontology.
- The modeling of documentation guidelines in the ontology.
- The generation of the conflict analysis report.

The time taken for all these steps has been measured separately.

A10 & B6: Evaluation of results

After the execution phase of the evaluation had been finished, it was necessary to extract the results and bring them into an easily comparable form. For the ontology-based approach it was primarily necessary to check the correctness of results, and the time taken. For the manual approach this included the analysis of each filled out form and check of the conflicts and categories that had been marked by the participants. The extraction of data has been

performed in a semi-automated way, by partly making use of MS Excel's automation capabilities.
In detail the data has been extracted from the following artifacts:

- 6 result sheets for requirements categorization from each of the 6 individual participants
- 6 result sheets for conflict analysis from each of the 6 individual participants
- 1 categorization result sheet from a requirement engineering expert
- 1 conflict analysis result sheet from a requirement engineering expert
- 1 categorization report created with OntRep
- 1 conflict analysis report created with OntRep
- 1 sheet on which the times measured for the ontology-based approach have been noted

The results were evaluated with statistical techniques using Excel and R and are described in the following section.

5 Evaluation Results

This section presents the results for the evaluation of ontology-based and manual requirements categorization and requirements conflict analysis.

5.1 Evaluation results of requirements categorization

Because there is no inherent “right” or “wrong” categorization of requirements, I chose the categorization results of the requirements engineering expert to serve as a reference. This means, that the more another result approximates the expert’s result, the more it is considered to be correct.

In the categorization process each requirement could be associated with an arbitrary number of categories: Zero, one or more than one. So I designed four classes of possible results:

- **Correct**
  “Correct” means that the related categories of the participant exactly matched the categories related by the expert.
  Example:
  Expert related the categories: 3,4,5
  Participant related the categories: 3,4,5
- **False**
  “False” means that none of the related categories of the participant matched the categories related by the expert.
  Example:
  Expert related the categories: 3,4,5
  Participant related the categories: 6

- **Partially correct**

  "Partially correct" means that one or more of the related categories of the participant matched the categories related by the expert, but not all of them.

  Example:

  Expert related the categories: 3,4,5

  Participant related the categories: 3,4

- **Overfulfilled**

  "Overfulfilled" means, that the related categories of the participant matched the categories related by the expert, but the participant also related more categories.

  Example:

  Expert related the categories: 3,4,5

  Participant related the categories: 3,4,5,6

Each result from the participants and also from the OntRep categorization was associated into one of these four classes. The following table shows the results ("P1" stands for "Participant 1" and "G1" stands for "Group 1"):

	Individual (average)	**Group (average)**	**OntRep**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**G1**	**G2**
overfulfilled	9,5	12,5	6	7	10	17	9	7	7	15	10
correct	5,7	6,0	8	7	5	1	9	6	6	4	8
partially correct	2,0	0,0	2	3	2	2	0	2	3	0	0
false	5,8	4,5	7	6	6	3	5	8	7	4	5

Table 1: Evaluation results of requirements categorization

To visualize these results I converted them into the following boxplots:

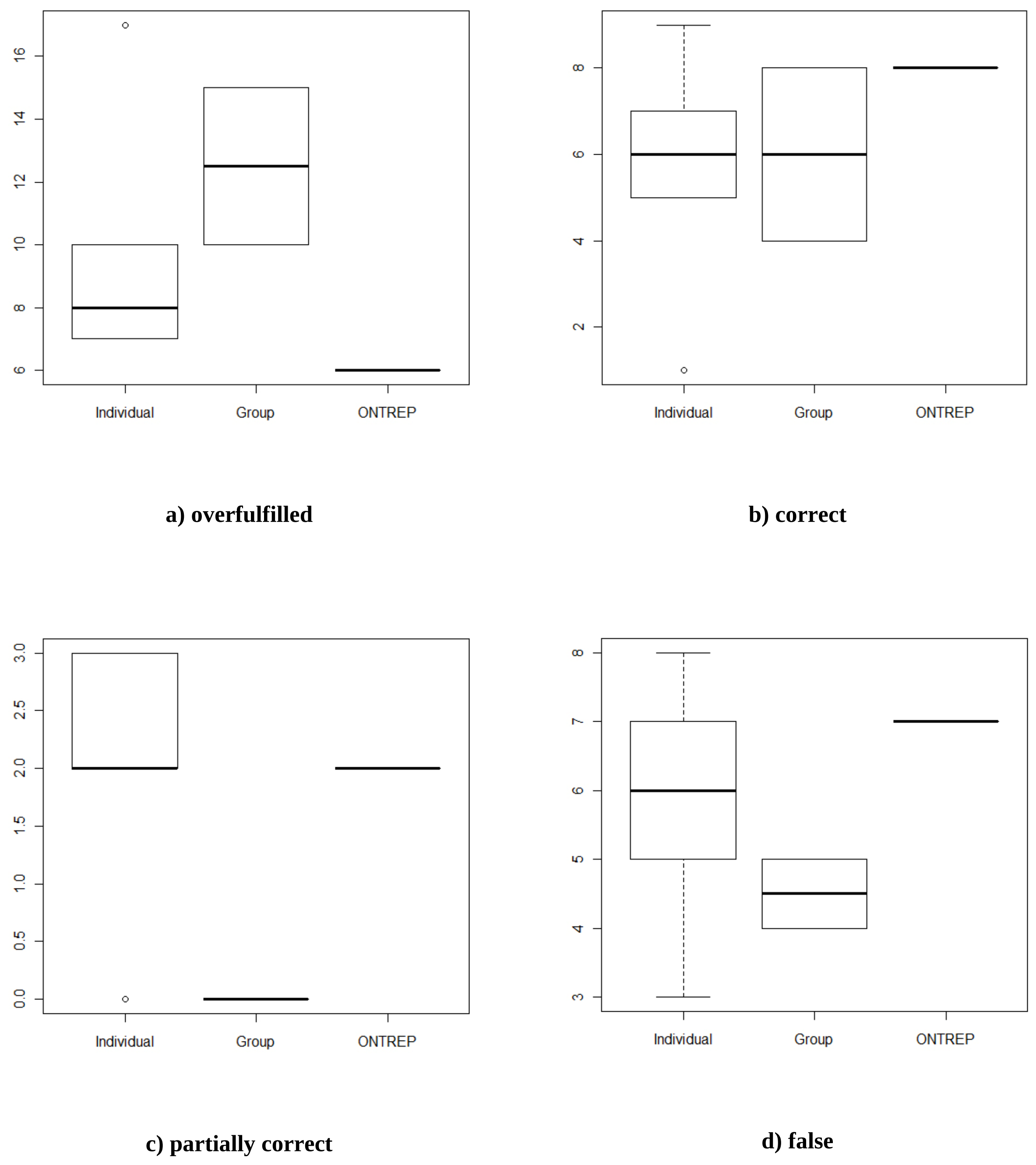


Figure 48: Boxplots visualizing the results of the requirements categorization

The results show that the group results could improve the individual results. The team reviews slightly increased the number of correct results, and clearly decreased the number of false categorizations. The number overfulfilled categorizations also increased, while partially correct results decreased. The overfulfillment of a categorization is not as severely wrong as a partially correct result. While an overfulfilled category only lowers the precision of the overall result, in a partially correct categorization some (maybe important) information is missing.

So the increase of overfulfilled and decrease of partially correct results in the group phase can be rated as an increase in correctness. Furthermore, it is a hint on how the teams had worked: The groups apparently tended to merge their individual results together rather than excluding poor results. This could have been caused through the collaborative nature of the office environment, where the participants possibly tend not to object to the opinions of others.

In comparison with the manual categorization, OntRep performed very well in terms of precision: OntRep had more correct results than the individuals on average and also than the groups on average. Looking at each single result in particular, there was only one participant (P4) that could achieve more correct results.

OntRep also did not overfulfill as much categorizations as the individuals and especially as the groups. This is another indicator for the high precision of the automated prototype.

The difference in partially correct results is not significant: OntRep did perform exactly as well as the individuals on average, and slightly worse as the groups did.

Great opportunities for further improvements exist regarding the false positive results: OntRep performed worse than the individuals and groups on average, and is among the worst results when comparing each single one in particular. The cause for this bad result were “empty” categorizations: 4 of the 7 false categorizations made by OntRep were requirements that have not been related to **any** category. Obviously, OntRep could not find any matching synonyms, hyponyms or substrings to relate those requirements to the right categories. A cause for this issue probably were the short texts of the requirements. But there is high potential to improve this result by investing more efforts into the user-defined specification of synonyms.

The following table shows the **efforts** that had to be invested in order to perform the requirements categorization:

	Individual (average)	**Group1 total**	**Group2 total**	**Group (avg. of totals)**	**OntRep**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**G1**	**G2**
Effort (in minutes)	15,33	95	66	80,5	16	5	24	18	10	16	19	16	7

Table 2: Efforts for requirements categorization

When analyzing the efforts, OntRep performed slightly worse than the individuals on average. The efforts of the groups must include the individual efforts of each participant (because the knowledge gained from there has been used in the team review), and add the time spent for the group review multiplied by three (because three persons had to invest their time in the meeting). So the total of group 1 has been calculated using the following formula:

$$P1+P2+P3+(G1*3)$$

The total efforts of the two groups on average are much higher than for OntRep. This indicates, that quality assurance and the collaborative improvement of results is very time-consuming.

But another aspect also needs to be analyzed when discussing efforts: Scalability.
The main effort that had to be invested in OntRep was the preparation of the ontology classes and the user-defined synonyms. The time for the processes that run automatically is insignificantly high at 2 minutes (possibly varying with the hardware used). It can be assumed that the number of categories (which cause the main effort) does not increase equally with the number of requirements, because a system with equal or more element classes than elements would reduce a classification to absurdity. This means that the overall effort for OntRep does not increase linearly with the number of requirements.
For the manual approach, each single requirement has to be categorized. This implies that the overall effort for the categorization increases with the number of requirements. Because every requirement has to be checked with every category, this effort increases more than linear.

The following graph visualizes an effort approximation with an increasing number of requirements:

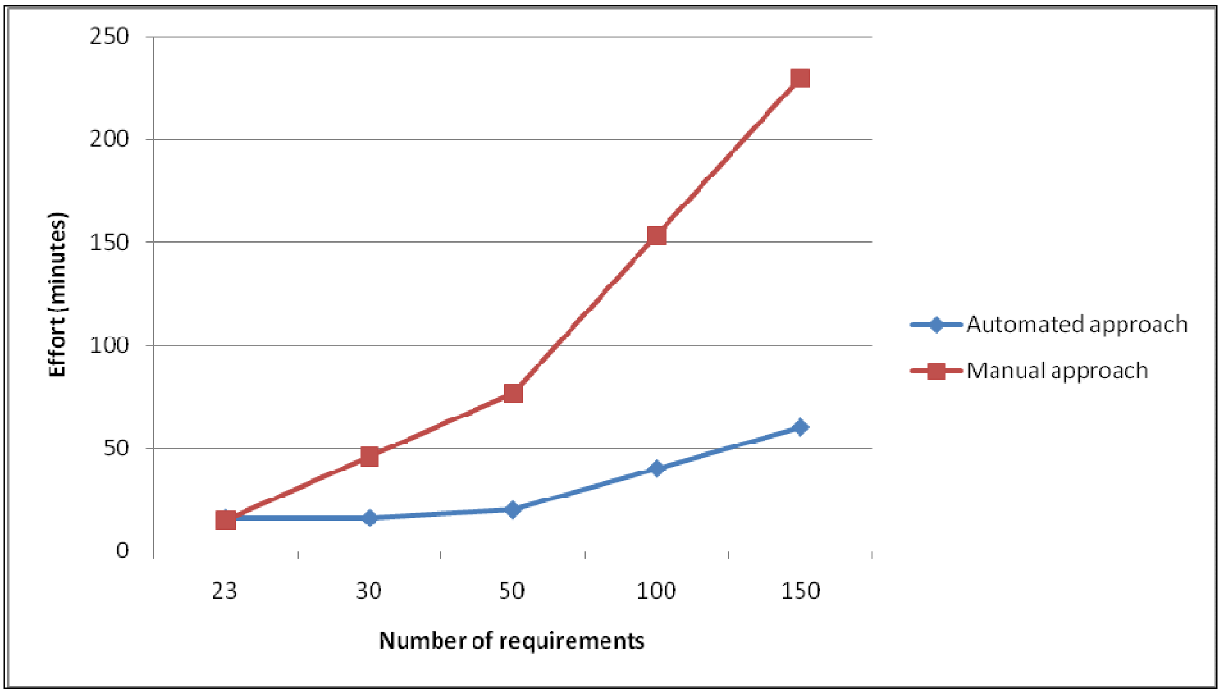


Figure 49: Effort approximation for categorization of an increasing number of requirements

The effort approximation assumes a 1:3 ratio of categories to requirements at a low number of requirements, and a 1:5 ratio at 50 and more requirements. This can be assumed due to the following fact: The more categories exist, the higher is the probability to be able to relate a new requirement into an already existing category.

The efforts for the automated approach in the approximation are based on the increasing number of categories (in a ratio based on the evaluation results), while the efforts of the manual approach are based on the increasing number of requirements (in a ratio based on the evaluation results).

The diagram shows, that the effort differences between automated and manual approach rapidly diverge with an increasing number of requirements.

Another aspect that has not been taken into account here and which was not topic of the evaluation is the **possibility of reuse**: When executing multiple software development projects in a similar context, there is the possibility of reusing the knowledge of an ontology (categories and synonyms), so the preparations would not need to start from zero. But this assumption needs further investigation.

5.2 Evaluation results of requirements conflict analysis

The second part of the evaluation was concerned with the finding of conflicts in the given test data. The reference solution consists of all conflicts in the test data that fall into one of the well-defined conflict classes described in section 3.3.2. Other possible conflicts have not been counted as true conflicts.
The reference solution has been built by myself by accurately analyzing the test data. It contains 22 true conflicts. Errors in this solution cannot be excluded by 100% probability. But because the automated result matched the reference solution exactly, there is a very high chance that the reference solution is correct.

Every conflict that had been marked by a participant has been classified into one of the following result categories:

- **Correctly identified conflicts**
  This class includes all true conflicts that have been marked by the participant, including all elements that are involved in the conflict (like for example glossary terms or technical constraints).
- **Correctly, but partially identified conflict**
  This class includes all true conflicts that have been marked by the participant, but without correctly marking all elements that are involved in the conflict (like for example glossary terms or technical constraints).
- **Conflicts not found**
  This class includes all true conflicts that have not been marked by the participant at all.
- **False positive conflicts**
  This class includes all conflicts that have been marked by the participant, but actually are **not** true conflicts.

Furthermore, two aggregated categories can be defined:

- **Sum of correctly identified conflicts**
  This is the sum of the correctly identified conflicts and the correctly, but only partially identified conflicts.
- **Total conflicts found**
  This includes all conflicts found by the participant, including false positive conflicts. It can be calculated by adding the sum of correctly identified conflicts to the false positive conflicts.

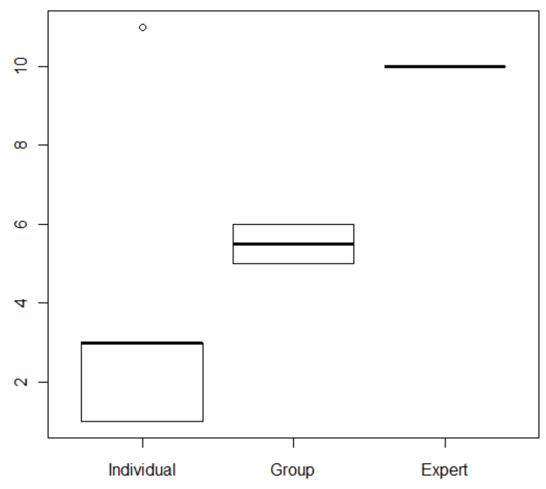
Each conflict from the participants and also from the OntRep report was associated with one of the described categories. OntRep correctly identified 100% of the true conflicts without false positives, so it won't be listed in the following comparisons.

The next table shows the absolute results of the manual conflict analysis ("P1" stands for "Participant 1" and "G1" stands for "Group 1"). The results of the requirements engineering expert are also included ("EXP"):

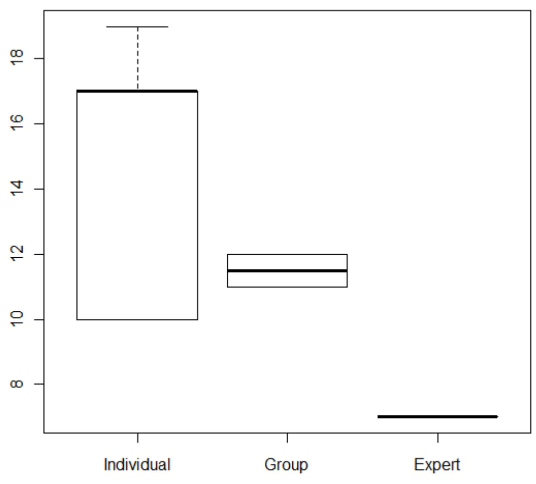
	Individual (average)	**Group (average)**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**G1**	**G2**	**EXP**
Correctly identified conflicts	3,66	5,5	3	3	1	3	1	11	5	6	10
Correctly, but partially identified Conflicts	3,33	5	9	2	4	2	2	1	6	4	5
Sum of correctly identified conflicts	7	10,5	12	5	5	5	3	12	11	10	15
Conflicts not found	15	11,5	10	17	17	17	19	10	11	12	7
False positive conflicts found	10	11	8	4	4	15	23	6	10	12	2
Total Conflicts found	17	21,5	20	9	9	20	26	18	21	22	17

Table 3: Absolute evaluation results of manual conflict analysis

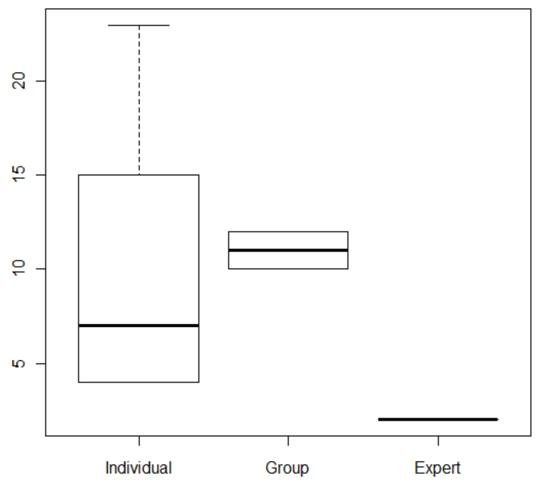
The visualization of these numbers produces the following boxplots:



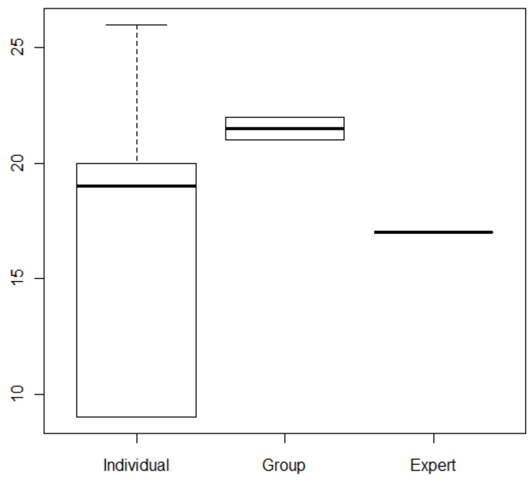
a) correctly identified conflicts



b) conflicts not found



c) false positive conflicts found



d) total conflicts found

Figure 50: The absolute results of the manual conflict analysis converted into boxplots

The following table shows the relative average results in percent. This answers the question: What percentage of the 22 true conflicts have been found?

The value for false positives states how many percent of the total conflicts that had been found were false positives.

	Relative average result (individual)	Relative average result (group)	Relative result (expert)
Correctly identified conflicts	16,67 %	25 %	45,45 %
Correctly, but partially identified Conflicts	15,15 %	22,73 %	22,73 %
Sum of correctly identified conflicts	31,82 %	47,73 %	68,18 %
Conflicts not found	68,18 %	52,27 %	31,82 %
False positive conflicts found	58,82 %	51,16 %	11,76 %

Table 4: Relative evaluation results of manual conflict analysis

From these results, the following conclusions can be drawn:

- As expected the reviews in the groups significantly improved the individual results. The number of identified conflicts increases from about a third to a half. The ratio of false positives has been slightly improved. But it can be seen from Table 3 that the absolute number of false positives has not been improved.

  Similar to the categorization, this indicates the behavior of the teams: The groups apparently tended to merge their individual results together rather than excluding poor results. It seems that they did not assess themselves as experienced enough to reject the conflicts found by others.
- Though the team reviews could improve the value for correctly identified conflicts, only about half of the true conflicts have been found in total (including partially identified conflicts). This value would probably not be satisfying in a real project environment, because lots of conflicts would cause problems at a later stage of the project, increasing their costs.
- The high number of false positives (more than half of the conflicts found were no true conflicts) is another indicator that the conflict analysis is a very challenging and error-prone task to perform manually. This is probably caused by the high number of elements that can be included in a conflict (different requirements, glossary terms, constraints).

  False positives are not as costly as conflicts that have not been found at all, because their correctness can be validated at an early stage of the project. But nevertheless, high efforts have to be invested into the inspection of the conflicts found.

- The requirements engineering expert achieved the best results. He correctly identified more conflicts at a lower error rate than the individuals and the groups. This shows that practice can significantly improve results, even more than team work.
  Nevertheless, the expert also did not find about a third of the true conflicts. An indicator that experience does not necessarily mean to achieve optimal results, and again highlights the major error-proneness of the task.

The following table shows the relative results split up into the different conflict classes. Overall there were 7 conflicts of type "CRC", 10 of type "CRG" and 5 of type "CRR". Only the most important numbers are shown:

	Relative average result (individual)	**Relative average result (group)**	**Relative result (expert)**
Conflict type "CRC"			
Sum of correctly identified conflicts	28,57 %	57,14 %	42,86 %
False positive conflicts found	68,42 %	50 %	0 %
Conflict type "CRG"			
Sum of correctly identified conflicts	30 %	50 %	80 %
False positive conflicts found	52,63 %	33,33 %	11,11 %
Conflict type "CRR"			
Sum of correctly identified conflicts	40 %	30 %	80 %
False positive conflicts found	53,85 %	75 %	20 %

Table 5: Relative average results by conflict type

The results show similar numbers for the conflicts of type "CRC" and "CRG". For both types, the team reviews could improve the overall correctness.
Conflict type "CRR" differs from that: Interestingly, here the team reviews worsened the individual results. Maybe this could have been caused by the different nature of "CRR" conflicts: While "CRC" and "CRG" are similar and therefore the teams had more practice with the finding of those conflicts, "CRR" conflicts are quite different. But it has to be said here that 5 conflicts of type "CRR" are not enough to be able to draw empirically proven conclusions.

Like the individuals the expert had most problems finding conflicts of type “CRC”. Furthermore, he did not mark any conflicts of that type wrong. This indicates that he was very careful with this type. Maybe he lacked practice at finding this particular conflict type. This shows that personal preferences and experience play an important role for conflict analysis.

The true conflicts can also be categorized by another attribute: Their complexity. I defined a conflict as “simple”, if at most 3 elements are involved in it (elements can be requirements, glossary terms, constraints). If 4 or more elements are involved, it is “complex” according to this definition.

The following table shows relative results according to conflict complexity. Overall there were 12 simple conflicts, and 10 complex conflicts to find.

The values for the false positives in the table state how many of the simple/complex conflicts marked by the participants (depending on the number of involved elements) were no true conflicts:

	Relative average result (individual)	**Relative average result (group)**	**Relative result (expert)**
Simple conflicts			
Sum of correctly identified conflicts	31,94 %	50 %	75 %
False positive conflicts found	67,61 %	53,85 %	18,18 %
Complex conflicts			
Sum of correctly identified conflicts	31,67 %	45 %	60 %
False positive conflicts found	36,67 %	47,06 %	0 %

Table 6: Relative average results by conflict complexity

The results are as expected: The individuals and the groups did not find as many complex as simple conflicts. Also, the results in the team reviews could be improved more for the simple than for the complex conflicts. The expert’s difference between simple and complex conflicts found is even more significant.

The table also shows that more of the false positive conflicts were simple than complex. This indicates, that the more elements a potential conflict involved, the more certain the participants had to feel about its correctness to really mark it.

The following table shows the **efforts** that had to be invested in order to perform the conflict analysis:

	Individual (average)	Group1 total	Group2 total	Group (avg. of totals)	OntRep	P1	P2	P3	P4	P5	P6	G1	G2	EXP
Effort (in minutes)	97,166	408	397	402,5	91	120	75	90	108	90	100	41	33	45

Table 7: Efforts for conflict analysis

Like it has also been done for requirements categorization, the efforts of the groups also have to consider the individual efforts of each participant (because the knowledge gained from there has been used in the team review). So the time spent for the group review multiplied by three (because three persons had to invest their time in the meeting) has to be added to the individual efforts of the group members. The total of group 1 has been calculated using the following formula:

P1+P2+P3+(G1*3)

In comparison, OntRep took slightly less effort than the individuals on average, and double the effort as the expert. Because the team reviews took the time of multiple persons, the total group efforts are about four times higher than for OntRep. This indicates that the execution of quality assurance is even more time-consuming for conflict analysis than it is for requirements categorization. Conflict analysis is a more complex task and this makes it harder to compare and review different results.

When looking at the aspect of **scalability**, we need to analyze what tasks made up the efforts for the OntRep conflict analysis:

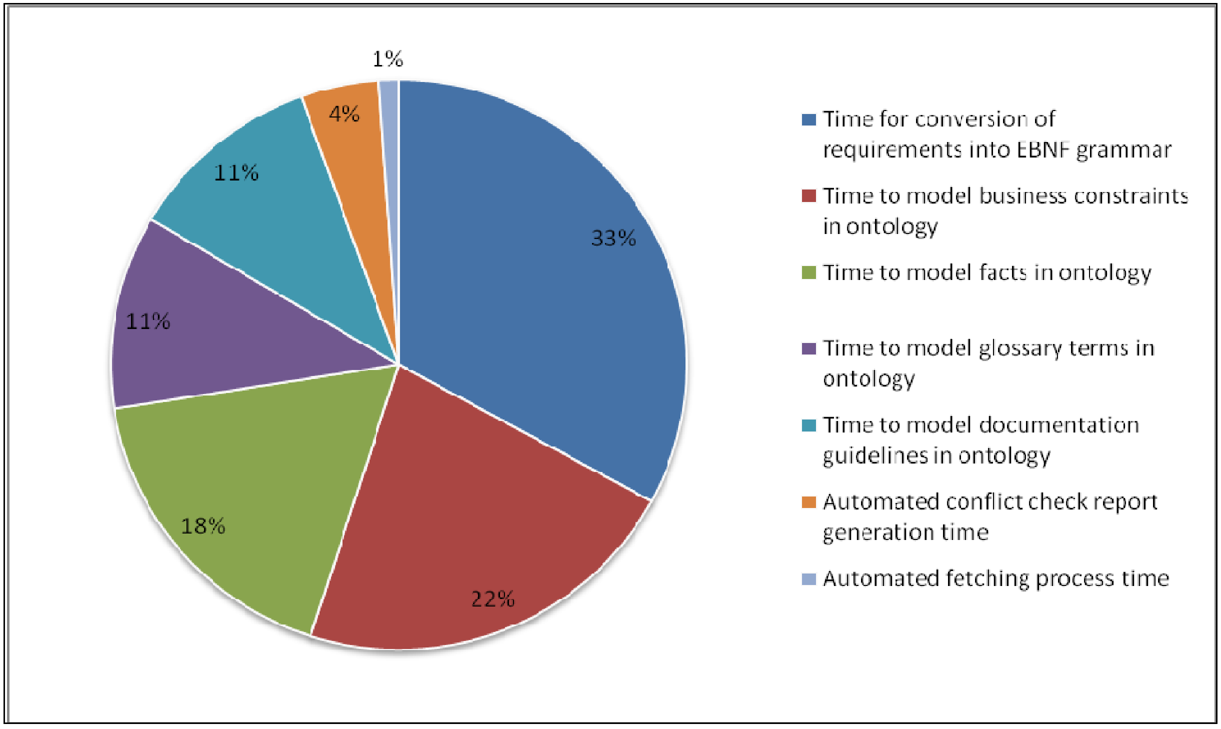


Figure 51: Particular efforts for automated conflict analysis

The diagram shows that most efforts had to be done for the task of converting the requirements into EBNF grammar, because this has to be done for each single requirement. The second most efforts took the modeling of elements in the ontology, especially the modeling of business constraints and relevant facts.

To approximate efforts for larger scales, I first counted all involved elements (requirements, constraints, guidelines, facts and glossary terms). Based on these numbers I assessed how many elements exist in a SRS with an increasing number of requirements, because with more requirements, also the number of facts, glossary terms and other elements increases.
To estimate the efforts for the preparation of the automated approach, I calculated how long it took for a single element in the evaluation setting, and multiplied it with the values of the increasing number of elements. Example: According to the evaluation, it took 30 minutes to convert 23 requirements into the EBNF grammar. So to convert 30 requirements, it takes approximately 39 minutes, to convert 50 requirements approximately 65 minutes,… and so on. I calculated values for each kind of element this way to get authentic values.

The approximation for the manual performance must include another aspect: For the automated approach, the preparation has to be done for each single element, without interdependence with other elements. But during the manual conflict analysis all the requirements have to be checked with each other element. So the manual efforts do not increase in a linear, but quadratic way.

To estimate the manual efforts, I approximated the number of relations that have to be checked with an increasing number of elements. We had 47 elements in our evaluation setting, thereof 23 requirements. So the 23 requirements had to be checked with 24 other elements, which results in 552 (23 multiplied by 24) relations to check.

According to the evaluation, it took the individuals 97 minutes to check the 552 relations. So to check 900 relations, it takes approximately 158 minutes, to check 2000 relations approximately 352 minutes,… and so on. I calculated the values for an increasing number of elements. Then I did the same for the expert, whose base value was 45 minutes for 552 relations. The following graph shows the results:

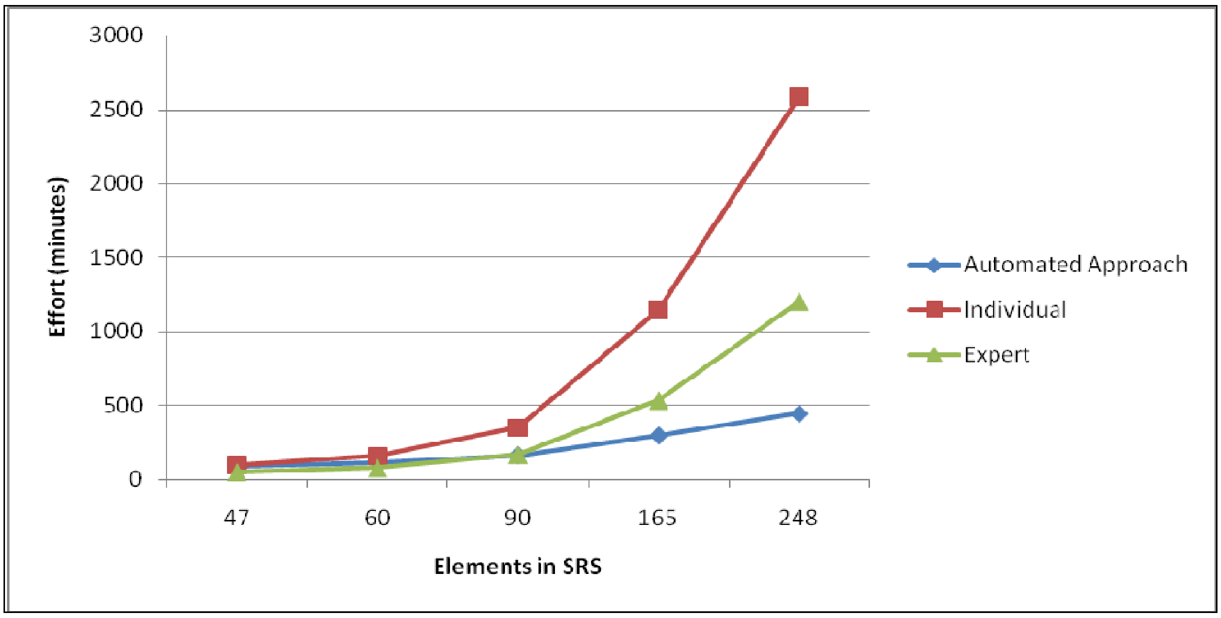


Figure 52: Approximating the efforts for conflict analysis for an increasing number of elements

The graph shows the sharp increase of efforts for individuals. The values start to differ significantly for 90 elements (which corresponds to 50 requirements), and skyrocket for 165 and 248 elements (which correspond to 100 and 150 requirements). Furthermore, these values do not consider any efforts for team reviews.

The efforts for the expert do not increase that fast, it is for 47 and 60 elements (which correspond to 23 and 30 requirements) even slightly lower than the automated approach. The expert’s effort meets the automated effort at 90 elements, and exceeds it at 165 and 248 elements.

When comparing these efforts, it always has to be considered that the automated approach provided the best results regarding conflict identification correctness.

5.3 Threats to validity

Several measurements were taken to assure the internal validity of the evaluation: The concept of the study was reviewed several times by different persons, an expert on how to conduct software engineering studies among them.
Furthermore, we executed a test run with a test person. This verified that the explanations and task descriptions of the evaluation forms are understandable for the participants, and helped to estimate the required time frame.

To assure the external validity of the evaluation, it was performed at the professional software development company Siemens Austria. The test data was based upon a real ongoing project at Siemens. The participants were all IT professionals, and had medium requirements engineering know-how and advanced software engineering know-how. In addition, the requirements engineering expert served as experimental “control group”.

Despite these measurements some potential threats might hinder the ability to generalize the evaluation results on a broad scale:

- The most obvious threat to the evaluation’s validity is the rather small number of participants. The number of 6 persons plus 1 expert does not allow to view the insights gained as empirically proven facts. Therefore, I suggest to rerun the study on a larger scale in future work.
- The efforts to converse the requirements into the specified EBNF grammar is highly dependent on the projects way of specifying requirements. A tool supported method could make any manual conversion unnecessary, while a specification document without any formal structure might multiply the estimated efforts.
- The number of conflicts in a software requirements specification depends on a multitude of factors: The number of constraints and guidelines, the accuracy of requirements, the process of requirements elicitation,… etc.

Therefore it is very difficult to make estimations of how many conflicts and constraints exist in the average requirements specification. The absolute conflict numbers of this evaluation shall not be generalized.

But in proportion, because a different number of conflicts or constraints influences the automated **and** the manual efforts, the evaluation results should be applicable to software specifications with more or less conflicts as well.

6 Discussion

The standard way of **categorizing requirements** in software development projects is to perform it manually. The results of the evaluation presented in this master thesis indicate that the automated approach OntRep is a proper alternative. The quality of OntRep's categorization was only slightly worse than manually performed, with similar efforts. OntRep uses existing information retrieval approaches that go beyond keyword matching to identify relationships between requirements and categories.

Because of the analysis of synonyms and hyponyms, OntRep can be used in domains where different stakeholders use different terminologies for the same concepts. This inconsistency makes the analysis of syntax-driven approaches limited and incomplete. Through WordNet, OntRep makes use of a large knowledge base where all concepts are linked semantically. Furthermore, it uses the benefits of ontologies and their expandable nature, to be able to map different terms to the same concept and thus support a better comparison of differently formulated requirements. Thus, the requirements engineers have more flexibility regarding the vocabulary to use for requirements formulation.

OntRep's categorization performance can be increased flexibly by investing more time into the specification of user-defined synonyms. This shall improve OntRep's handicap of not being able to relate all requirements to at least one category.

The big advantage of OntRep lies in its scalability: According to approximations based on the evaluation results, the effort for ontology preparations does increase much less than the effort for manually categorizing a high number of requirements. But it needs further research to confirm this.

There are different ways of how to define a **conflict** between requirements depending on the context, but in general it is a particular kind of interdependency between different elements. In terms of conflict analysis, there is a range of different techniques reported in literature, e.g. code overlaps [Egyed, Grünbacher, 2004], or by defining a specific logic [Hunter, Nuseibeh, 1997]. Generally there is the distinction between formal and informal, and between syntactic and semantic approaches (also see section 2.5). Because OntRep analyzes the contents of the requirements and semantically processes their texts, it can be called a formal, semantic approach.

OntRep's approach for conflict analysis is based on ontologies. Their graph-based structure of elements and reasoning capabilities are well-suited for this kind of task.
I focused the evaluation on three conflict types: Conflicts between requirements and business constraints, conflicts between requirements and documentation guidelines, and numerical conflicts between requirements (possibly involving technical constraints). The choice for these types of conflict was a reasonable trade-off between covering a broad range of possible conflicts and tool implementation efforts. Especially the automated check for conflicts of type "CRR" holds the potential to be extended to find more than only numerical conflicts.

The results of the evaluation of the automated conflict analysis are promising. The evaluation was designed to not only consider the manual conflict analysis of individuals, but also included group work to simulate review meetings. The feedback from the participants was very clear: One stated that it was "hard to do the conflict analysis", another one wrote "it is difficult and cumbersome to find the conflicts". The statistics extracted from the evaluation results speak the same language: The high efforts that had been invested by the participants resulted only in a small number of correctly identified conflicts.
Of course OntRep's conflict analysis does also come with efforts: The biggest preparation task according to the evaluation is the conversion of the requirements into the specified EBNF grammar. One could argue that most requirements are specified in plain text in the majority of today's software requirements specification documents. On the other hand, the importance of a specified grammar to use for requirements increases when requirements databases are used, which is typical for large projects. Because there, the requirements need to be clear and understandable even without the context from a document. Furthermore, the usage of requirements management tools can support the user in defining the requirements in valid grammar.
For OntRep the way of modeling the ontology is of major importance: It can only find conflicts involving facts or constraints that have been modeled correctly in the ontology. The constraints work like an exactly specified filter on the requirements. This also implies, that if the user did not specify them in the intended way, the tool cannot find the corresponding conflicts. If a constraint does not exist in the ontology at all, OntRep cannot find any conflicts related to that constraint. And if a constraint has been modeled wrongly, it might find conflicts with that constraint it was not supposed to find. But if all facts and constraints have been modeled correctly, OntRep finds all corresponding conflicts.

In comparison, the results of the evaluation showed that OntRep brings results of much higher quality than when performing the conflict analysis manually. The preparation efforts were similar to the manual efforts of persons with basic requirements engineering know-how, and higher than for the requirements engineering expert for a low number of elements. But OntRep’s results regarding scalability look promising: Approximations show that its preparation efforts do only increase linearly, while the manual efforts increase much faster, even non-linearly.

The concept still has room for improvements, but seems to be a promising new approach.

7 Conclusion & Further Work

Software engineering projects are becoming more complex nowadays: This is caused by an increasing number of requirements, increasing complexity of their contents and relationships among them, and project members with different domain backgrounds and terminologies. To keep an overview on the requirements, project managers and requirements engineers relate them to different categories. To prevent costly changes in the requirements at a later stage of the project, they perform a conflict analysis on the requirements and existing constraints. However, the manual performance of these tasks is a time-consuming work and very error-prone.

In this master theses I proposed an approach based on semantic technologies for automating these requirements engineering tasks and introduced the automated ontology-based tool "OntRep", which makes usage of the ontology description language OWL-DL and semantic reasoning mechanisms. The requirements need to be formulated in a grammar I specified using EBNF, which supports the automated analyzing of the requirements contents.

I evaluated the costs and the benefits of the ontology-based approach by conducting a real-world industrial case study at Siemens Austria with 6 project managers divided into 2 teams. The study focused on the comparison of automated and manual performance of the tasks regarding efforts and results quality. In the evaluation, the participants had to manually categorize the requirements of the case study project into a given set of categories and analyze the requirements for conflicts between them and constraints. In addition, a requirements engineering expert was assigned the same tasks. The OntRep tool automatically performed the categorization and conflict analysis on the same data. I measured the necessary efforts and the quality of results for all participants and for OntRep.

The evaluation results showed that OntRep can be a proper alternative or amendment to the manual categorization of requirements in typical software development projects, because it provides results of only slightly lower quality with similar efforts, but is scalable much higher. Its benefit increases with the number of requirements to categorize.

OntRep's categorization quality can be increased by investing efforts into the definition of synonyms to the ontology.

In the conflict analysis, OntRep identified all true conflicts in the requirements, while manual approaches only found 50% to 60% and produced much higher rates of false positives with similar efforts. OntRep can analyze three conflict types at the moment: Conflicts between requirements and business constraints, between requirements and documentation guidelines, and numerical conflicts between requirements (involving technical constraints). Approximations show that OntRep is much more scalable than the manual approach.

The ontology-based approach seems to be helpful for project managers, who want to manage the requirements with less effort, but keep their consistency high. Both tasks can be conducted by using OntRep. The management in software development projects benefits from reduced manual effort for requirements categorization and conflict analysis, and reduced communication and clarification effort through automated conflict analysis support.

OntRep can be improved in **further work** by supporting the user in modeling the constraints and the facts of the ontology. At the moment, all modeling has to be done manually. Also, the conversion of requirements into the specified EBNF grammar can be performed much more efficiently through tool support.
The types of conflicts that can be found by OntRep shall be extended in future work. Especially the type "CRR" can be expanded to include more than only numerical conflicts between requirements. The ontology-based approach could include the possibility to freely define conflict types manually with semantic means in the future. This would allow to extend the conflict types to be found without restrictions.
OntRep could also be extended to automatically find semantic relationships between different requirements. Example relationships could be "describes", "consists of" or "depends on". At the moment, it is only possible to define these relationships manually.

Further work shall repeat the evaluation in a larger environment, with more participants to improve the validity of results. In addition, the number of requirements and constraints shall be increased in order to analyze the correctness of the approximations made, and gain results of higher scale.

References

[Grothe, 1999] Martin Grothe 1999. “Aufbau von Business Intelligence, Entwicklung einer softwaregestützten Controlling-Kompetenz bei o.tel.o“. In krp Kostenrechnungspraxis

[Chamoni and Gluchowski, 2006] P. Chamoni, P. Gluchowski 2006. “Analytische Informationssysteme Einordnung und Überblick.“ In: P. Chamoni, P. Gluchowski: „Analytische Informationssysteme. Business Intelligence Technologien.“
2. edition. Springer, 2006, page 2 – 22

[ProClarity, 2005] ProClarity July 2005. “Blending Reporting and Analytics: Putting the DecisionMaker First.“

[Kemper et. al., 2006] Hans-Georg Kemper, Walid Mehanna and Carsten Unger 2006. “Business intelligence Grundlagen und praktische Anwendungen ; eine Einführung in die IT-basierte Managementunterstützung“. 2. edition, Vieweg+Teubner Verlag, ISBN 3834802751 (ISBN13: 9783834802750)

[Spahn et. al., 2008] Michael Spahn, Joachim Kleb, Stephan Grimm and Stefan Scheidl 2008. „Supporting Business Intelligence by Providing Ontology-Based End-User Information Self-Service“. In ACM International Conference Proceeding Series Vol. 308, Proceedings of the first international workshop on Ontology-supported business intelligence

[Gruber, 1995] T. Gruber 1995. “Toward Principles for the Design of Ontologies Used for Knowledge Sharing.” In Formal Ontology in Conceptual Analysis and Knowledge Representation, N. Guarino and R. Poli (eds), Kluwer, Academic Publishers

[Gruber, 1993] Thomas R. Gruber 1993. “A Translation Approach to Portable Ontology Specifications“. In Knowledge Acquisition, 5(2):199-220.

[Pedrinaci et. al., 2008] C. Pedrinaci, D. Lambert, B. Wetzstein, T. van Lessen, L. Cekov, and M. Dimitrov 2008. “SENTINEL: a semantic business process monitoring tool.” In Proceedings of the First international Workshop on ontology-Supported Business intelligence (Karlsruhe, Germany, October 27 - 27, 2008). OBI '08, vol. 308. ACM, New York, NY, 1-12.

[Jena] http://jena.sourceforge.net/ontology/index.html

[Hitzler, 2007] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, York Sure. „Semantic Web: Grundlagen“ (eXamen.press). Springer, 1. edition, October 2007. ISBN 978-3-540-33993-9

[RDF] http://www.w3.org/RDF/

[Turtle] http://www.w3.org/TeamSubmission/turtle

[RDF-S] http://www.w3.org/TR/rdf-schema/

[OWL] http://www.w3.org/TR/owl-semantics/

[MKS] MKS Whitepaper 2009. „Echte Wiederverwendung von Anforderungen“. www.mks.com

[Gluchowski, 2006] P. Gluchowski, H.-G. Kemper 2006. “Quo Vadis Business Intelligence?”. In BI-Spektrum, issue 1, 2006, page 12–19

[Mikroyannidis et. al., 2006] A. Mikroyannidis, B. Theodoulidis, and A. Persidis 2006. “PARMENIDES: Towards Business Intelligence Discovery from Web Data“. In Proceedings of the 2006 IEEE/WIC/ACM international Conference on Web intelligence (December 18 - 22, 2006). Web Intelligence. IEEE Computer Society, Washington, DC, 1057-1060.

[Mohania, Bhide, 2008] M. Mohania and M. Bhide 2008. “New trends in information integration.“ In Proceedings of the 2nd international Conference on Ubiquitous information

Management and Communication (Suwon, Korea, January 31 - February 01, 2008). ICUIMC '08. ACM, New York, NY, 74-81.

[Kohavi et. al., 2002] R. Kohavi. N.J. Rothleder and E. Simoudis 2002. “Emerging trends in business analytics”. Commun. ACM 45, 8 (Aug. 2002), 45-48.

[Vega, 2001] C. A. Vega 2001. “Java and reports, some solutions for the past, present and future.” In Proceedings of the 29th Annual ACM SIGUCCS Conference on User Services (Portland, Oregon, USA, October 17 - 20, 2001). SIGUCCS '01, vol. 29. ACM, New York, NY, 275-278.

[Jinfonet] http://www.jinfonet.com/

[Tomcat] http://tomcat.apache.org/

[JSP] http://java.sun.com/products/jsp/index.jsp

[Keller, 2004] Patrick Keller, Business Application Research Center (BARC), “Markttrends Berichts- und Analysewerkzeuge”, from magazine “Monitor” 10/2004

[Kleijn, 2006] Alexandra Kleijn 08.06.2006. „Business Intelligence mit Open Source“ from heise Open Source. http://www.heise.de/open/Business-Intelligence-mit-Open-Source--/artikel/73725

[JasperReports] http://jasperreports.sourceforge.net/

[Pentaho] http://www.pentaho.com/products/reporting/try_reporting.php

[Actuate] http://www.actuate.com/home/index.asp

[Lenzerini, 2002] M. Lenzerini 2002. “Data integration: a theoretical perspective”. In Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (Madison, Wisconsin, June 03 - 05, 2002). PODS '02. ACM, New York, NY, 233-246.

[Wache et. al., 2001] Wache, Vogele, Visser, Stuckenschmidt, Schuster, Neumann and Hubner 2001. “Ontology-based integration of information - a survey of existing approaches”. In IJCAI-01 Workshop: Ontologies and Information Sharing

[Calvanese et al., 2001] Diego Calvanese, Giuseppe De Giacomo and Maurizio Lenzerini 2001. “Description logics for information integration”. In Computational Logic: From Logic Programming into the Future (In honour of Bob Kowalski), Lecture Notes in Computer Science. Springer-Verlag

[Goasdoué et al., 1999] Francois Goasdoué, Véronique Lattes and Marie-Christine Rousset 1999. “The use of carin language and algorithms for information integration: The picsel project”. In International Journal of Cooperative Information Systems (IJCIS), 9(4):383 – 401

[Goh, 1997] Cheng Hian Goh 1997. “Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Sources.” Phd, MIT.

[Saggion et. al., 2007] Horacio Saggion, Adam Funk, Diana Maynard and Kalina Bontcheva 2007. “Ontology based information extraction for business intelligence.” In Proc. ISWC

[Lamsweerde et. al., 1998] Axel van Lamsweerde, Robert Darimont, Emmanuel Letier 1998. "Managing Conflicts in Goal-Driven Requirements Engineering". In IEEE Transactions on Software Engineering, vol. 24, no. 11, pp. 908-926, Nov. 1998

[Hunter, Nuseibeh, 1997] A. Hunter and B. Nuseibeh 1997. “Analyzing Inconsistent Specifications”. In Proceedings of the 3rd IEEE international Symposium on Requirements Engineering (January 05 - 08, 1997). RE. IEEE Computer Society, Washington, DC, 78.

[Egyed, Grünbacher, 2004] A. Egyed, P. Grünbacher 2004. “Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help”. IEEE software

[Heitmeyer et. al., 1995] C.L. Heitmeyer, R.D. Jeffords, B.G. Labaw 1995. "Automated consistency checking of requirements specifications". 2nd International Symposium on Requirements Engineering (RE '95), York, England

[Choi, 2000] F.Y.Y. Choi 2000. "Advances in domain independent linear text segmentation". In Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference., Morgan Kaufmann Publishers Inc., Seattle, Washington

[WordNet, 1995] G. A. Miller 1995. "WordNet: A Lexical Database for English". In Communications of the ACM Vol. 38, No. 11: 39-41

[Chitchyan et. al., 2006] Chitchyan, Sampaio, Rashid and Rayson 2006. "A tool suite for aspect-oriented requirements engineering". In Proceedings of the 2006 international Workshop on Early Aspects At ICSE (Shanghai, China, May 21 - 21, 2006). EA '06. ACM, New York, NY, 19-26

[Sampaio et. al., 2005] Sampaio, Chitchyan, Rashid and Rayson 2005. "EA-Miner: a tool for automating aspect-oriented requirements identification". In Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering (Long Beach, CA, USA, November 07 - 11, 2005). ASE '05. ACM, New York, NY, 352-355

[Trac] http://trac.edgewall.org/

[SPARQL] http://www.w3.org/TR/rdf-sparql-query/

[Stopp-words] http://www.textfixer.com/resources/common-english-words.php

[Porter-Stemmer] http://tartarus.org/~martin/PorterStemmer/

[EBNF] http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

[Rupp, 2004] Chris Rupp 2004. "Requirements Engineering und –Management" Hanser, 3. edition

[Python]	http://www.python.org/
[Genshi]	http://genshi.edgewall.org/
[JPype]	http://jpype.sourceforge.net/
[pywordnet]	http://osteele.com/projects/pywordnet/
[Protegé]	http://protege.stanford.edu/overview/protege-owl.html
[Pellet]	http://clarkparsia.com/pellet/

[ViSEK, 2002] B. Freimut, T. Punter, St. Biffl, M. Ciolkowski 2002. "State-of-the-Art in Empirical Studies". In Report: ViSEK/007/E, Fraunhofer Inst. of Experimental Software Engineering